

2024 개정 내용 완벽 반영

# SQLD

## 완벽 이면 정리

무단 전재 및 재배포를 금지합니다.



HDATA LAB | 대표강사 홍은혜

# 이 책의 목차

## 1과목 : 데이터 모델링의 이해

### PART1. 데이터 모델링의 이해

1-1. 데이터 모델의 이해	1	1-4. 관계	10
1-2 엔터티(Entity)	4	1-5. 식별자	11
1-3. 속성	7		

### PART2. 데이터 모델과 SQL

1-6. 정규화	15	1-9. Null 속성의 이해	23
1-7. 관계와 조인의 이해	19	1-10. 본질식별자 vs 인조식별자	26
1-8. 모델이 표현하는 트랜잭션의 이해	22		

## 2과목 : SQL 기본 및 활용

### PART1. SQL 기본

2-1. 관계형 데이터베이스 개요	29	2-5. GROUP BY, HAVING 절	48
2-2. SELECT 문	32	2-6. ORDER BY 절	50
2-3. 함수	36	2-7. 조인	54
2-4. WHERE 절	43	2-8. 표준 조인	61

### PART2. SQL 활용

2-9. 서브 쿼리	67	2-13. Top N 쿼리	102
2-10. 집합 연산자	78	2-14. 계층형 질의와 셀프 조인	108
2-11. 그룹 함수	83	2-15. PIVOT 절과 UNPIVOT 절	112
2-12. 윈도우 함수	90	2-16. 정규 표현식	117

### PART3. 관리 구문

2-17. DML	126	2-19. DDL	133
2-18. TCL	131	2-20. DCL	150

# 1과목. 데이터 모델링의 이해

## 1과목 출제과목 변동(2024년 기준)

변경 전		변경 후	
주요항목	세부항목	주요항목	세부항목
데이터 모델링의 이해	<ul style="list-style-type: none"> <li>데이터모델의 이해</li> <li>엔터티</li> <li>속성</li> <li>관계</li> <li>식별자</li> </ul>	데이터 모델링의 이해	<ul style="list-style-type: none"> <li>데이터모델의 이해</li> <li>엔터티</li> <li>속성</li> <li>관계</li> <li>식별자</li> </ul>
데이터 모델과 성능	<ul style="list-style-type: none"> <li>정규화와 성능</li> <li>반정규화와 성능</li> <li>대용량 데이터에 따른 성능</li> <li>DB 구조와 성능</li> <li>분산 DB 데이터에 따른 성능</li> </ul>	데이터 모델과 SQL	<ul style="list-style-type: none"> <li>정규화</li> <li>관계와 조인의 이해</li> <li>모델이 표현하는 트랜잭션의 이해</li> <li>Null 속성의 이해</li> <li>본질식별자 vs 인조식별자</li> </ul>

출처 : 한국데이터산업진흥원

엔터티(Entity), 속성(Attribute), 인스턴스(Instance)에 대해서는 아래와 같이 생각하면 쉽습니다.

<학생테이블>					
학번	이름	전화번호	주소	학과번호	지도교수번호
1001	홍길동	010-0000-0000	서울시 ...	101	10000
1001	박길동	010-1111-1111	경기도 ...	102	10001
...	...	...	...	...	...

컬럼(속성)  
행(인스턴스)  
테이블(엔터티)

## 1-1. 데이터 모델의 이해

### ● 모델링의 개념

- 현실 세계의 비즈니스 프로세스와 데이터 요구 사항을 추상적이고 구조화된 형태로 표현하는 과정
- 데이터베이스의 구조와 관계를 정의하며, 이를 통해 데이터의 저장, 조작, 관리 방법을 명확하게 정의

### ● 모델링의 특징

1. 단순화(Simplification)
  - 현실을 단순화하여 핵심 요소에 집중하고 불필요한 세부 사항을 제거
  - 단순화를 통해 복잡한 현실 세계를 이해하고 표현하기 쉬워짐

## 2. 추상화(Abstraction)

- 현실세계를 일정한 형식에 맞추어 간략하게 대략적으로 표현하는 과정
- 다양한 현상을 일정한 양식인 표기법에 따라 표현

## 3. 명확화(Clarity)

- 대상에 대한 애매모호함을 최대한 제거하고 정확하게 현상을 기술하는 과정
- 명확화를 통해 모델을 이해하는 이들의 의사소통을 원활히 함

## ● 데이터 모델링 3 가지 관점

### 1. 데이터 관점

- 데이터가 어떻게 저장되고, 접근되고, 관리되는지를 정의하는 단계

### 2. 프로세스 관점

- 시스템이 어떤 작업을 수행하며, 이러한 작업들이 어떻게 조직되고 조정되는지를 정의하는 단계
- 데이터가 시스템 내에서 어떻게 흐르고 변환되는지에 대한 확인

### 3. 데이터와 프로세스 관점

- 데이터 관점과 프로세스 관점을 결합하여 시스템의 전반적인 동작을 이해하는 단계
- 특정 프로세스가 어떤 데이터를 사용하는지, 데이터가 어떻게 생성되고 변경되는지를 명확하게 정의

## ● 데이터 모델링 유의점

### 1. 중복(Duplication)

- 한 테이블 또는 여러 테이블에 같은 정보를 저장하지 않도록 설계

### 2. 비유연성(Inflexibility)

- 사소한 업무 변화에 대해서도 잣은 모델 변경이 되지 않도록 주의
- 데이터 정의를 프로세스와 분리

### 3. 비일관성(Inconsistency)

- 데이터베이스 내의 정보가 모순되거나 상반된 내용을 갖는 상태를 의미
- 데이터간 상호연관 관계를 명확히 정의
- 데이터 품질 관리 필요
- 데이터의 중복이 없더라도 비일관성은 발생할 수 있음

## ● 데이터 모델링 3 가지 요소

- 대상(Entity) : 업무가 관리하고자 하는 대상(객체)
- 속성(Attribute) : 대상들이 갖는 속성(하나의 특징으로 정의될 수 있는 것)
- 관계(Relationship) : 대상들 간의 관계

## ● 데이터 모델링의 3 단계

### 1. 개념적 모델링

- 업무 중심적이고 포괄적(전사적)인 수준의 모델링
- **추상화 수준이 가장 높음**
- **업무를 분석 뒤 업무의 핵심 엔터티(Entity)를 추출하는 단계**
- 도출된 핵심 엔터티(Entity)들과의 관계들을 표현하기 위해 ERD 작성

### 2. 논리적 모델링

- 개념적 모델링의 결과를 토대로 세부속성, 식별자, 관계 등을 표현하는 단계
- 데이터 구조를 정의하기 때문에 비슷한 업무나 프로젝트에서 동일한 형태의 데이터 사용 시 재사용 가능
- 동일한 논리적 모델을 사용하는 경우 쿼리도 재사용 가능
- 데이터 정규화 수행
- 재사용성이 높은 논리적 모델은 유지보수가 용이해짐

### 3. 물리적 모델링

- **논리 모델링이 끝나면 이를 직접 물리적으로 생성하는 과정**
- 데이터베이스 성능, 디스크 저장구조, 하드웨어의 보안성, 가용성 등을 고려
- 가장 구체적인 데이터 모델링
- **추상화 수준은 가장 낮음(가장 구체적인 모델링이므로)**

## ● 스키마의 3 단계 구조

- 스키마 : 데이터베이스의 구조와 제약 조건에 관한 전반적인 명세를 기술한 메타데이터의 집합
- 외부, 개념, 내부 스키마로 분리
- 사용자의 관점과 실제 설계된 물리적인 방식을 분리하기 위해 고안됨

### 1. 외부 스키마

- **사용자가 보는 관점**에서 데이터베이스 스키마를 정의
- 사용자나 응용 프로그램이 필요한 데이터를 정의(View : 사용자가 접근하는 대상)

### 2. 개념 스키마

- 사용자 관점의 데이터베이스 스키마를 통합하여 **데이터베이스의 전체 논리적 구조를 정의**
- 전체 데이터베이스의 개체, 속성, 관계, 데이터 탑입 등을 정의

### 3. 내부 스키마

- 데이터가 **물리적으로 어떻게 저장되는지를 정의**
- 데이터의 저장 구조, 컬럼, 인덱스 등을 정의함

## \*\* 3 단계 스키마의 독립성

- 독립성 : 물리적, 논리적 구조를 변경하더라도 사용자가 사용하는 응용 프로그램에 영향을 주지 말아야 함
- 1) 논리적 독립성 : 논리적 데이터 구조가 변경되어도(개념 스키마 변경) 응용 프로그램에 영향을 주지 않는 특성
- 2) 물리적 독립성 : 물리적 구조가 변경되어도(내부 스키마 변경) 개념/외부 스키마에 영향을 주지 않는 특성

## ● 데이터 모델의 표기법(ERD : Entity Relationship Diagram)

- 엔터티(Entity)와 엔터티 간의 관계(Relationship)를 시각적으로 표현한 다이어그램
- 1976년 피터 첸(Peter Chen)이 만든 표기법, 데이터 모델링 표준으로 사용

## ● ERD 작성 절차 (6 단계)

- ① 엔터티를 도출한 후 그린다
- ② 엔터티 배치
- ③ 엔터티 간의 관계를 설정
- ④ 관계명을 서술
- ⑤ 관계의 참여도 기술
- ⑥ 관계의 필수 여부를 확인

## 1-2. 엔터티

### ● 엔터티(Entity)의 개념

- 현실 세계에서 독립적으로 식별 가능한 객체나 사물을 나타냄
- 엔터티는 업무상 분석해야 하는 대상(Instance)들로 이루어진 집합
- 인스턴스는 엔터티의 특정한 속성 값들로 구성되며, 엔터티의 개념을 현실에서 구체적으로 나타낸 것  
예) 엔터티와 속성, 인스턴스 등의 관계

- 엔터티(Entity) : 학생
- 속성(Attribute) : 학번, 이름, 학과 등.
- 식별자(Identifier) : 학번 (고유한 학번으로 각 학생을 식별)
- 인스턴스 : 특정 학생의 데이터
  - 학번: 2021001
  - 이름: 흥길동
  - 학과: 컴퓨터 공학

### ● 엔터티(entity)의 특징

1. 유일한 식별자에 의해 식별 가능
  - 인스턴스가 식별자에 의해 한 개씩만 존재하는지 검증 필요
  - 유일한 식별자는 그 엔터티의 인스턴스만의 고유 이름  
ex) 이름은 동명이인이 있을 수 있으므로 사번, 학번 등이 고유식별자

## 2. 해당 업무에 필요하고 관리하고자 하는 정보

- 설계하는 업무의 시스템 구축에 필요한 정보이어야 함  
ex) 학교 시스템 구축 시 학생정보 필요. 다른 업무엔 학생 정보 불필요.

## 3. 인스턴스들의 집합

- 영속적으로 존재하는 2 개 이상의 인스턴스의 집합
- 인스턴스가 한 개 밖에 없는 엔터티는 집합이 아니므로 성립이 안됨.

## 4. 엔터티는 반드시 속성을 가짐

- 각 엔터티는 2 개 이상의 속성을 가짐
- 하나의 인스턴스는 각각의 속성들에 대한 1 개의 속성 값만을 가짐  
ex) 학생 엔터티에서 한 학생의 데이터(인스턴스)의 이름(속성) 정보에는 반드시 한 값만 저장됨

## 5. 엔터티는 업무 프로세스에 의해 이용

- 업무적으로 필요해 선정했지만 실제 사용되지 않으면 잘못 설계된 것
- 모델링 시 발견하기 어려운 경우 데이터 모델 검증이나 상관 모델링 시 단위 프로세스 교차점검으로 문제 도출
- 누락된 프로세스의 경우 추후 해당 프로세스 추가
- 반대로 사용되지 않는 고립 엔터티는 제거 필요

## 6. 다른 엔터티와 최소 1 개 이상의 관계 성립

- 엔터티는 업무적 연관성을 갖고 다른 엔터티와 연관의 의미를 가짐
- 관계가 없는 엔터티 도출은 부적절한 엔터티이거나 적절한 관계를 찾지 못한 것

## ● 엔터티의 분류

### 1) 유형과 무형에 따른 분류

#### ① 유형엔터티

- 물리적 형태가 있음(실체가 있는 대상)
- 안정적이며 지속적으로 활용되는 엔터티
- 업무로부터 구분하기가 가장 용이한 엔터티  
ex) 사원, 물품, 강사 등

#### ② 개념엔터티

- 물리적인 형태 없음
- 관리해야 할 개념적 정보로부터 구분되는 엔터티  
ex) 조직, 보험상품 등

#### ③ 사건엔터티

- 업무를 수행에 따라 발생하는 엔터티
- 발생량이 많고 각종 통계자료에 이용  
ex) 주문, 청구, 미납 등

## 2) 발생 시점에 따른 분류

### ① 기본엔터티

- 그 업무에 원래 존재하는 정보
- 다른 엔터티와 관계에 의해 생성되지 않고 독립적으로 생성
- 타 엔터티의 부모 역할을 하는 엔터티
- 다른 엔터티로부터 주식별자를 상속받지 않고 자신의 고유한 주식별자를 가짐
- ex) 사원, 부서, 고객, 상품 등

### ② 중심엔터티

- 기본엔터티로부터 발생되고 그 업무에서 중심적인 역할
- 많은 데이터가 발생되고 다른 엔터티와의 관계를 통해 많은 행위 엔터티를 생성
- ex) 계약, 사고, 청구, 주문, 매출 등

### ③ 행위엔터티

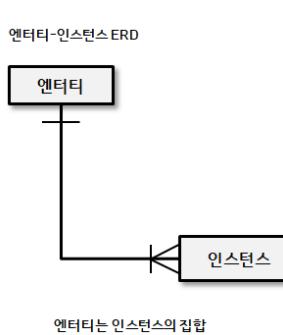
- 2개 이상의 부모엔터티로부터 발생
- 자주 내용이 바뀌거나 데이터 양이 증가
- 분석 초기 단계보다는 상세 설계 단계나 프로세스와 상관모델링을 진행하면서 도출
- ex) 주문(고객과 상품 엔터티로 부터 발생하므로 행위엔터티이기도 함), 사원변경이력, 이력 등

## ● 엔터티의 명명

- ① 현업에서 사용하는 용어 사용
- ② 가능하면 약자 사용은 자제
- ③ 단수 명사 사용
- ④ 모든 엔터티에서 유일하게 이름 부여
- ⑤ 엔터티 생성 의미대로 이름 부여

## ● 엔터티와 인스턴스 표기법

- 엔터티는 사각형으로 표현, 속성은 조금씩 다름



엔터티-인스턴스 예

엔터티	인스턴스
과 목	수 학
	영 어
강 사	김 쌤
	홍 쌤
강 의 실	자연관
	본관

< 엔터티와 인스턴스 >

[IE 표기법]

과목	강사	강의실 PK
과목이름	강사이름	강의실이름
...	...	...

[Baker 표기법]

과목 PK  
 # 과목이름  
 ...  
 ○ ...

강사  
 # 강사이름  
 ...  
 ○ ...

강의실  
 # 강의실이름  
 ...  
 ○ ...

< 엔터티 표기법 >

## 1-3. 속성

### ● 속성(Attribute)의 개념

- 속성은 업무에서 필요로 하는 고유한 성질, 특징을 의미(관찰 대상) -> 컬럼으로 표현할 수 있는 단위!
- 업무상 인스턴스로 관리하고자 하는 더 이상 분리되지 않는 최소의 데이터 단위
- 인스턴스의 구성 요소
- ex) 학생 엔터티에 이름, 학번, 학과번호 등이 속성이 될 수 있음

### ● 엔터티, 인스턴스, 속성, 속성값의 관계

- 한 개의 엔터티는 2개 이상의 인스턴스의 집합이어야 한다(하나의 테이블은 두 개 이상의 행을 가짐)
- 한 개의 엔터티는 2개 이상의 속성을 갖는다(하나의 테이블은 두 개 이상의 컬럼으로 구성됨)
- 한 개의 속성은 1개의 속성값을 갖는다(각 컬럼의 값은 하나씩만 삽입 가능)
- 속성은 엔터티에 속한 엔터티에 대한 자세하고 구체적인 정보를 나타냄, 각 속성은 구체적인 값을 가짐

### ● 속성의 특징

- 반드시 해당 업무에서 필요하고 관리하고자 하는 정보여야 한다
- 정해진 주식별자에 함수적 종속성을 가져야 한다
- 하나의 속성은 한 개의 값만을 가진다.(한 컬럼의 값은 각 인스턴스마다 하나씩만 저장)
- 하나의 속성에 여러 개의 값이 있는 다중값일 경우 별도의 엔터티를 이용하여 분리한다
- 하나의 인스턴스는 속성마다 반드시 하나의 속성값을 가진다
- => 각 속성이 하나의 값을 갖고 있음을 의미(속성의 원자성)

#### \*\* 원자성이란

- 데이터모델에서 각 엔터티의 인스턴스가 해당 속성에 대해 단일하고 명확한 값을 가지는 것을 의미

### ● 함수적 종속성

- 한 속성의 값이 다른 속성의 값에 종속적인 관계를 갖는 특징을 말함
- 즉, 어떤 속성 A의 값에 의해 다른 속성 B도 유일하게 결정된다면, B는 A에 함수적으로 종속됐다 하고, 이를 수식으로 나타내면  $A \rightarrow B$  라고 표현함

#### 1) 완전 함수적 종속

- 특정 컬럼이 기본키에 대해 완전히 종속될 때를 말함
- PK를 구성하는 컬럼이 2개 이상일 경우 PK 값 모두에 의한 종속관계를 나타낼 때 완전 함수 종속성 만족
- ex) (주문번호 + 제품번호)에 의해 수량 컬럼의 값이 결정됨

주문		PK
주문번호	제품번호	수량
1	100	4
1	101	1
2	100	3
2	101	2

## 2) 부분 함수적 종속

- 기본키 전체가 아니라, 기본키 일부에 대해 종속될 때를 말함

ex) 수강기록 테이블에서 학생번호와 과목이 PK라고 가정할 때, 과목에 의해서도 교수가 결정되면 부분 함수적 종속 관계!

수강기록		PK
학생번호	과목	강사
1001	수학	홍쌤
1002	수학	홍쌤
1001	과학	최쌤
1003	영어	김쌤

## ● 속성의 분류

### 1) 속성의 특성에 따른 분류

#### ① 기본 속성

- 업무로부터 추출된 모든 속성
  - 엔터티에 가장 일반적으로 많이 존재하는 속성
- ex) 원금, 예치기간 등

#### ② 설계 속성

- 기본 속성 외에 업무를 규칙화하기 위해 새로 만들어지거나 기본 속성을 변형하여 만들어지는 속성
- ex) 상품코드, 지점코드, 예금분류 등

#### ③ 파생 속성

- 다른 속성에 의해 만들어지는 속성
  - 일반적으로 계산된 값들이 해당
  - 데이터 정합성을 유지하기 위해 가급적 적게 정의하는 것이 좋음
- ex) 합계, 평균, 이자 등

### 2) 엔터티 구성방식에 따른 분류

#### ① PK(Primary Key, 기본키)

- 인스턴스를 식별할 수 있는 속성

#### ② FK(Foreign Key, 외래키) 속성

- 다른 엔터티와의 관계에서 포함된 속성

③ 일반 속성

- 엔터티에 포함되어 있고 PK/FK에 포함되지 않는 속성

**3) 분해 여부에 따른 속성**

① 단일 속성

- 하나의 의미로 구성된 경우
- ex) 회원 ID, 이름 등

② 복합 속성

- 여러개의 의미로 구성된 경우
- ex) 주소(시, 구, 동 등으로 분해 가능) 등

③ 다중값 속성

- 속성에 여러 개의 값을 가질 수 있는 경우
- 다중값 속성은 엔터티로 분해
- ex) 상품 리스트 등

● 속성의 명명규칙

- ① 해당 업무에서 사용하는 이름을 부여
- ② 서술식 속성명은 사용하지 않음
- ③ 약어의 사용은 가급적 제한
- ④ 전체 데이터 모델에서 유일한 명칭

● 도메인(Domain)

- 도메인은 각 속성이 가질 수 있는 값의 범위를 의미함
- 엔터티 내에서 속성에 대한 데이터 타입과 크기, 제약사항을 지정하는 것이다

## 1-4. 관계

### ● 관계(Relationship)의 개념

- **관계는 엔터티간의 연관성을 나타낸 개념**
- 관계를 정의할 때는 인스턴스(각 행 데이터)간의 논리적인 연관성을 파악하여 정의
- 엔터티를 어떻게 정의하느냐에 따라 변경되기도 함

### ● 관계의 종류

#### 1) 존재적 관계

- 한 엔터티의 존재가 다른 엔터티의 존재에 영향을 미치는 관계
- 엔터티 간의 연관된 상태를 의미
- ex) 부서 엔터티가 삭제되면 사원 엔터티의 존재에 영향을 미침

#### 2) 행위적 관계

- 엔터티 간의 어떤 행위가 있는 것을 의미
- ex) 고객 엔터티의 행동에 의해 주문 엔터티가 발생

\* ERD에서는 존재관계와 행위관계를 구분하지 않는다.

### ● 관계의 구성

1. 관계명
2. 차수(Cardinality)
3. 선택성(Optionality)

### ● 관계의 차수 (Cardinality)

- **한 엔터티의 레코드(인스턴스)가 다른 엔터티의 레코드(인스턴스)와 어떻게 연결되는지를 나타내는 표현**
- 주로 1:1, 1:N, N:M 등으로 표현

#### 1) 1 대 1 관계

- **완전 1 대 1 관계**
  - 하나의 엔터티에 관계되는 엔터티가 반드시 하나로 존재하는 경우
  - ex) 사원은 반드시 소속 부서가 있어야 함
- **선택적 1 대 1 관계**
  - 하나의 엔터티에 관계되는 엔터티가 하나이거나 없을 수 있는 경우
  - ex) 사원은 하나의 소속 부서가 있거나 아직 발령전이면 없을 수 있음

## 2) 1 대 N 관계

- 엔터티에 하나의 행에 다른 엔터티의 값이 여러 개 있는 관계
- ex) 고객은 여러 개 계좌를 소유할 수 있음.

## 3) N 대 M 관계

- 두 엔터티가 다대다의 연결 관계 가지고 있음
- 이 경우 조인 시 카테시안 곱이 발생하므로 두 엔터티를 연결하는 연결엔터티의 추가로 1 대 N 관계로 해소할 필요가 있음
- ex) 한 학생이 여러 강의를 수강할 수 있고, 한 강의 기준으로도 여러 학생이 보유할 수 있음  
=> 이 두 엔터티의 연결엔터티로는 구매이력 엔터티가 필요함

### ● 관계의 페어링

- 엔터티 안에 인스턴스가 개별적으로 관계를 가지는 것
- 관계란 페어링의 집합을 의미함

#### \*\* 관계와 차수, 페어링 차이

- 학생과 강의 엔터티는 관계를 가짐
- 한 학생은 여러 강의를 수강할 수 있고, 한 강의도 여러 학생에게 수강될 수 있으므로 M 대 N 관계이며, 이 때 차수는 M:N 가 됨
- 인스턴스의 관계를 보면 "학생 A 가 강의 B 를 2023 년 1 학기에 수강했고 성적은 'A+'를 받았다"와 같은 특정한 페어링이 형성
- 이런식으로 관계의 차수는 하나의 엔터티와 다른 엔터티 간의 레코드 연결 방식을 나타내는 반면, 관계의 페어링은 두 엔터티 간의 특정 연결을 설명하고 추가 정보를 제공하는 용도로 사용.

## 1-5. 식별자

### ● 식별자 개념

- 하나의 엔터티에 구성된 여러 개의 속성 중에 엔터티를 대표할 수 있는 속성을 나타냄
- 하나의 유일한 식별자가 존재해야 함
- 식별자는 논리 모델링에서 사용하는 용어, 물리 모델링에서는 키(key)라고 표현
- ex) 학생 엔터티의 주식별자는 학생번호 속성 => 학생 테이블의 기본키는 학생번호 컬럼  
(논리 모델링) (물리 모델링)

### ● 주식별자 특징

- ① 유일성 : 주식별자에 의해 모든 인스턴스를 유일하게 구분함

ex) 학생 엔터티에서 이름 속성은 동명이인이 발생할 수 있으므로 모든 인스턴스를 완벽하게 구분할 수 없으므로 학생번호와 같은 유일한 식별자를 주식별자로 사용

② 최소성 : 주식별자를 구성하는 속성은 유일성을 만족하는 최소한의 속성으로 구성

ex) 학생 엔터티의 주식별자는 학생번호만으로 충분한데, 학생번호 + 이름으로 구성할 필요없음

③ 불변성: 주식별자가 한번 특정 엔터티에 지정되면 그 식별자의 값은 변하지 않아야 함

(항상 고유값으로 존재해야 함)

ex) 학생 엔터티에 주식별자인 학생번호가 때에 따라 변경되어서는 안됨

④ 존재성 : 주식별자가 지정되면 반드시 값이 존재해야 하며 NULL은 허용 안 됨

## ● 식별자 분류

1) 대표성 여부에 따른 식별자의 종류

주식별자	보조식별자
<ul style="list-style-type: none"> <li>유일성과 최소성을 만족하면서 엔터티를 대표하는 식별자</li> <li>엔터티 내에서 각 인스턴스를 유일하게 구분할 수 있는 식별자</li> <li>타 엔터티와 참조관계를 연결할 수 있는 식별자</li> </ul>	<ul style="list-style-type: none"> <li>엔터티 내에서 각 인스턴스를 구분할 수 있는 구분자지만, 대표성을 가지지 못해 참조관계를 연결을 할 수 없는 식별자</li> <li>유일성과 최소성은 만족하지만 대표성을 만족하지 못하는 식별자</li> </ul>

2) 생성 여부에 따른 식별자의 종류

내부식별자	외부식별자
<ul style="list-style-type: none"> <li>다른 엔터티 참조 없이 엔터티 내부에서 스스로 생성되는 식별자</li> </ul>	<ul style="list-style-type: none"> <li>다른 엔터티와 관계로 인하여 만들어지는 식별자 (외래키)</li> </ul>

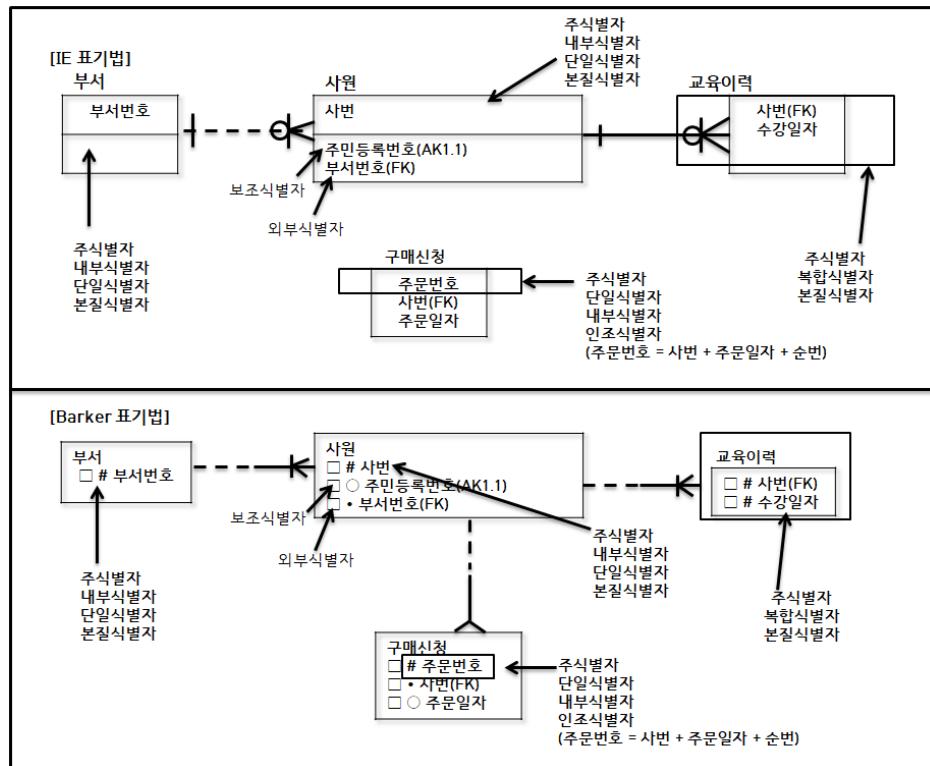
3) 속성 수에 따른 식별자 종류

단일식별자	복합식별자
<ul style="list-style-type: none"> <li>하나의 속성으로 구성</li> </ul>	<ul style="list-style-type: none"> <li>2 개 이상의 속성으로 구성</li> </ul>

4) 대체 여부에 따른 식별자의 종류

본질식별자(원조식별자)	인조식별자
<ul style="list-style-type: none"> <li>비즈니스 프로세스에서 만들어지는 식별자</li> </ul>	<ul style="list-style-type: none"> <li>인위적으로 만들어지는 식별자</li> <li>자동 증가하는 일련번호 같은 형태</li> </ul>

## ● 식별자 표기법



## ● 주식별자 도출기준

- 1) 해당 업무에서 자주 이용되는 속성을 주식별자로 지정한다.
  - 같은 식별자 조건을 만족하더라도 업무적으로 더 많이 사용되는 속성을 주식별자로 지정
  - ex) 학생번호와 주민번호 중에 학생번호가 주식별자, 주민번호는 보조식별자
- 2) 명칭이나 내역 등과 같은 이름은 피함
  - 이름 자체를 주식별자로 사용하는 행위를 피함
  - ex) (부서코드가 정의되지 않은 경우) 부서명 보다는 부서코드를 인위적으로 생성하여 부서코드를 주식별자로 사용
- 3) 속성의 수를 최대한 적게 구성
  - 주식별자를 너무 많은 속성으로 구성 시, 조인으로 인한 성능저하 발생 우려
  - 일반적으로 7~8 개 이상의 주식별자 구성은 새로운 인조식별자를 생성하여 모델을 단순화 시키는 것이 좋음
  - ex) 주문 엔터티에 대해 주문일자 + 주문상품코드 + 고객번호 + .... 등으로 구성 => 주문번호 속성 추가!

## ● 관계간 엔터티 구분

### 1) 강한 개체

- 독립적으로 존재할 수 있는 엔터티
- ex) 고객과 계좌 엔터티 중, 고객은 독립적으로 존재할 수 있음

### 2) 약한 개체

- 독립적으로 존재할 수 없는 엔터티
- ex) 고객과 계좌 엔터티 중, 계좌는 독립적으로 존재할 수 없음(고객에 의해 파생되는 엔터티)

## ● 식별 관계와 비식별관계

### 1) 식별관계(Identification Relationship)

- 하나의 엔터티가 자신의 기본키를 다른 엔터티가 기본키의 하나로 공유하는 관계
- 식별관계는 ERD에서 실선으로 표시
- ex) 사원과 교육이력 엔터티에서 양쪽 모두 기본키 중 일부가 사원번호임

사원	교육이력
#사원번호	#사원번호(FK)
	#수강일자

### 2) 비식별관계(Non-identification Relationship)

- 강한 개체의 기본키를 다른 엔터티에서 기본키가 아닌 일반 속성으로 관계를 가지는 것
- 비식별관계는 ERD에서 점선으로 표시
- ex) 부서와 사원의 관계에서 부서의 부서번호(기본키)를 사원 엔터티에서는 일반키로 가짐  
(사원에서는 사원번호가 기본키)

## ● Key의 종류

- 논리 모델링에서의 식별자가 물리 모델링에서는 Key가 되는데, 이를 Key의 특징에 따라 다음과 같이 분류

### ① 기본키(Primary Key)

- 엔터티를 대표할 수 있는 키

### ② 후보키(Candidate Key)

- 유일성(유니크한 특성)과 최소성(최소한의 컬럼으로 유일성을 만족하는 특징)을 만족하는 키
- 결국 후보키들 중 하나가 기본키가 되고, 나머지를 대체키라고 부름

### ③ 슈퍼키(Super Key)

- 유일성은 만족하지만 최소성은 만족하지 않는 키
- ex) 학생 테이블에서 학번으로만 PK를 구성해도 되는데, (학번 + 이름)으로 구성한다면 이는 슈퍼키임

#### ④ 대체키(Alternate Key)

- 여러 후보키 중 기본키가 아닌 키

#### ⑤ 외래키(Foreign Key)

- 다른 테이블의 기본키를 참조하는 키(다른 테이블의 기본키에 존재하는 값만 입력될 수 있는 특성을 갖는 키)
- 참조 테이블은 하나 또는 여러 개 가능

## 1-6. 정규화

모델링 시 최대한 중복 데이터를 허용하지 않아야 저장공간의 효율적 사용과 업무 프로세스의 성능을 기대할 수 있다. 이러한 중복 데이터를 허용하지 않는 방식으로 테이블을 설계하는 방식을 정규화라고 한다.

### ● 정규화(DB Normalization)의 개념

- 하나에 엔터티에 많은 속성을 넣게 되면, 해당 엔터티를 조회할 때마다 많은 양의 데이터가 조회될 것이므로 최소한의 데이터만을 하나의 엔터티에 넣는식으로 데이터를 분해하는 과정을 정규화라고 한다
- 데이터의 일관성, 최소한의 데이터 중복, 최대한의 데이터 유연성 위한 과정이라고 볼 수 있음
- 데이터의 중복을 제거하고 데이터 모델의 독립성을 확보
- 데이터 이상현상을 줄이기 위한 데이터 베이스 설계 기법
- 엔터티를 상세화하는 과정으로 논리 데이터 모델링 수행 시점에서 고려됨
- 제 1 정규화부터 제 5 정규화까지 존재, 실질적으로는 제 3 정규화까지만 수행

### ● 이상현상(Abnormality)

- 정규화를 하지 않아 발생하는 현상(삽입이상, 갱신이상, 삭제이상)
- 특정 인스턴스가 삽입 될 때 정의되지 않아도 될 속성까지도 반드시 입력되어야 하는(삽입이상) 현상이 발생함  
ex) 만약 사원 + 부서 엔터티를 합쳐 놓고 사원번호, 사원이름, 전화번호, 부서번호, 부서명, 부서위치의 속성이 존재할 때 새로운 사원 값이 추가될 때 정해지지 않은 부서정보(부서번호, 부서명, 부서위치) 모두 임의값 또는 NULL이 삽입되어야 함. 반대로 부서가 새로 추가 될 경우 소속 사원이 없어도 사원과 관련된 모든 속성이 불필요하게 값이 입력되어야 함
- 불필요 한 값까지 입력해야 되는 현상을 삽입이상, 그 외 갱신이상, 삭제이상이 발생할 수 있음  
ex) 부서 정보만 삭제하면 되는데 관련된 사원 정보까지도 함께 삭제되는 현상(삭제이상)

## ● 정규화 단계

### 1. 제 1 정규화(1NF)

- 테이블의 컬럼이 원자성(한 속성이 하나의 값을 갖는 특성)을 갖도록 테이블을 분해하는 단계
- 쉽게 말해 하나의 행과 컬럼의 값이 반드시 한 값만 입력되도록 행을 분리하는 단계

#### 예제) 구매 테이블의 제 1 정규화

상품에 여러 값이 있으므로 이를 여러 인스턴스로 분해



☞ 홍길동과 박길동은 구매상품이 두 값이 입력되어 있으므로 이를 각각 두 행으로 분리하는 작업을 거쳐야 함!

### 2. 제 2 정규화(2NF)

- 제 1 정규화를 진행한 테이블에 대해 완전 함수 종속을 만들도록 테이블을 분해
- 완전 함수 종속이란, 기본키를 구성하는 모든 컬럼의 값이 다른 컬럼을 결정짓는 상태
- 기본키의 부분 집합이 다른 컬럼과 1:1 대응 관계를 갖지 않는 상태를 의미
- 즉, PK(Primary Key)가 2 개 이상일 때 발생하며 PK의 일부와 종속되는 관계가 있다면 분리한다.

#### 예제) 수강이력 테이블의 제 2 정규화

기본키(학생번호 + 강의명)중, 강의명에 의해 강의실이 결정 -> 완전 함수 종속성 위배

(부분 함수 종속성을 가짐) -> PK 와 부분 함수 종속성을 갖는 컬럼을 각각 다른 테이블로 분해!

수강이력			
학생번호	강의명	강의실	성적
1000	컴퓨터공학	자연관	A
1001	컴퓨터공학	자연관	B
1002	기초통계	본관	A+
1000	데이터베이스	자연관	B+
1001	경영학	본관	C
1003	경영학	본관	A



수강이력		
학생번호	강의명	성적
1000	컴퓨터공학	A
1001	컴퓨터공학	B
1002	기초통계	A+
1000	데이터베이스	B+
1001	경영학	C
1003	경영학	A

강의실	
강의명	강의실
컴퓨터공학	자연관
기초통계	본관
데이터베이스	자연관
경영학	본관

☞ 수강이력에서는 한 학생이 여러 강의를 수강할 수 있기 때문에 주식별자는 학생번호로만는 불가능(유일성 불만족 때문) 따라서 학생번호와 강의명과 결합되어 주식별자가 되어야 한다. (한 학생이 같은 강의는 수강할 수 없다고 가정) 이 때, 주식별자의 부분집합인 **강의명에 의해 강의실이 달라지는 1 대 1 대응관계**를 갖는것을 완전 함수 종속성 위배, 같은 말로 부분 함수 종속 관계라고 하는데. 제 2 정규화는 이러한 부분 함수 종속성을 깨는 것을 목표로 한다. 따라서 주식별자를 분리할 수 없으니 주식별자는 수강이력에 그대로 있고, 문제가 되는 강의실 컬럼을 주식별자와 분리!

### 3. 제 3 정규화(3NF)

- 제 2 정규화를 진행한 테이블에 대해 이행적 종속을 없애도록 테이블을 분리
- 이행적 종속성이란  $A \rightarrow B, B \rightarrow C$  의 관계가 성립할 때,  $A \rightarrow C$  가 성립되는 것을 말함
- (A,B)와 (B,C)로 분리하는 것이 제 3 정규화

#### 예제) 구매 테이블 제 3 정규화

고객번호에 의해 상품명이 결정, 상품명에 의해 가격이 결정되는데  
 고객번호에 의해서도 구매 가격이 결정됨(고객이 상품을 결정하면 그에 매칭되는 가격이 결정되는 구조이므로)  
 따라서 (고객번호 + 상품명)과 (상품명 + 가격)으로 분리하는 것이 제 3 정규화!

구매		
고객번호	상품명	가격
1001	우유	2500
1002	치즈	3500
1003	소시지	4000
1004	우유	2500



구매	
고객번호	상품명
1001	우유
1002	치즈
1003	소시지
1004	우유

상품	
상품명	가격
우유	2500
치즈	3500
소시지	4000

☞ 이 경우 테이블을 분리하지 않으면, 구매 테이블에서 상품명을 변경해야 하는 상황이 발생할 경우 그 때마다 구매 테이블에서도 가격을 변경해야 한다. 하지만 제 3 정규화를 진행하여 테이블을 분리하게 되면, 구매 테이블에서의 상품명만 변경하면 되므로 업데이트에 비효율성이 줄어든다!

### 예제) 학생 테이블의 제 3 정규화

학번은 과목의 결정자이며, 과목은 교수의 결정자이다. 이 때, 학번이 달라지면 그 학번에 의한 교수가 달라지므로 학번 역시 교수의 결정자라고 얘기 할 수 있다. 따라서 전공과 교수 컬럼을 분리해야 함. 학생 테이블에서 교수정보가 삭제되고, 따로 과목테이블이 생기면서 교수의 결정자인 전공과 함께 들어간다.

학생		
학번	전공	교수
1001	통계학	홍교수
1002	수학	김교수
1003	경제학	최교수
1004	경영학	박교수



학생	
학번	전공
1001	통계학
1002	수학
1003	경제학
1004	경영학

과목	
전공	교수
통계학	홍교수
수학	김교수
경제학	최교수
경영학	박교수

### 예제) 계좌번호 제 3 정규화

계좌 테이블(분리 전)에서 계좌번호가 관리점코드의 결정자이며, 관리점코드 역시 관리점의 결정자인 상태에서 계좌번호에 의해 관리점도 달라지므로 계좌번호 역시 관리점에 대한 결정자이다. 이 때는 PK 외 두 속성을 분리, 따라서 관리점이 계좌 테이블에서 삭제되고, 따로 관리점 테이블로 분리되면서 이의 결정자인 관리점코드가 따라감

계좌			
계좌번호	예수금	관리점코드	관리점
100111	100	1000	서울점
100222	200	1001	경기점
100333	300	1002	인천점
100444	400	1003	제주점



계좌		
계좌번호	예수금	관리점코드
100111	100	1000
100222	200	1001
100333	300	1002
100444	400	1003

관리점	
관리점코드	관리점
1000	서울점
1001	경기점
1002	인천점
1003	제주점

### \* 결정자와 종속관계

만약 A 속성이 B 속성의 값을 결정하게 되면, 이 때 A는 B의 결정자라고 하며, 반대로 B는 A에 종속된다 표현함. 따라서 위 예제에서는 고객번호가 상품명을 결정자이며, 상품명 역시 가격의 결정자이다.

## 4. BCNF(Boyce-Codd Normal Form) 정규화

- 모든 결정자가 후보키가 되도록 테이블을 분해하는 것(결정자가 후보키가 아닌 다른 컬럼에 종속되면 안됨)

## 5. 제 4 정규화

- 여러 컬럼들이 하나의 컬럼을 종속시키는 경우 분해하여 다중 값 종속성을 제거

## 6. 제 5 정규화

- 조인에 의해서 종속성이 발생되는 경우 분해

### ● 반정규화 = 역정규화(De-Normalization)의 개념

- 데이터베이스의 성능 향상을 위해 데이터 중복을 허용하고 조인을 줄이는 데이터베이스 성능 향상 방법
  - 시스템의 성능 향상, 개발 및 운영의 단순화를 위해 정규화된 데이터 모델을 중복, 통합, 분리하는 데이터 모델링 기법
  - 조회(SELECT) 속도를 향상시키지만, 데이터 모델의 유연성은 낮아짐
- ※ 비정규화는 정규화를 수행하지 않음을 의미

### ● 반정규화 수행 케이스

- 정규화에 충실하여 종속성, 활용성을 향상되지만 수행 속도가 느려지는 경우
- 다량의 범위를 자주 처리해야 하는 경우
- 특정 범위의 데이터만 자주 처리하는 경우
- 요약/집계 정보가 자주 요구되는 경우

## 1-7. 관계와 조인의 이해

### ● 관계(Relationship)의 개념

- 엔터티의 인스턴스 사이의 논리적인 연관성
- 엔터티의 정의, 속성 정의 및 관계 정의에 따라서도 다양하게 변할 수 있음
- 관계를 맺는다는 의미는 부모의 식별자를 자식에 상속하고, 상속된 속성을 맵핑키(조인키)로 활용  
-> 부모, 자식을 연결함

사원 테이블			부서 테이블		
사번	이름	부서번호	부서번호	부서명	위치
1000	홍길동	100	100	영업부	서울
1001	최길동	101	101	관리부	서울
1002	김길동	201	201	생산부	부산

### ● 관계의 분류

- 관계는 존재에 의한 관계와 행위에 의한 관계로 분류
- **존재 관계는 엔터티 간의 상태를 의미**  
ex) 사원 엔터티는 부서 엔터티에 소속

- 행위 관계는 엔터티 간의 어떤 행위가 있는 것을 의미

ex) 주문은 고객이 주문할 때 발생

## ● 조인의 의미

- 결국 데이터의 충복을 피하기 위해 테이블은 정규화에 의해 분리된다. 분리되면서 두 테이블은 서로 관계를 맺게 되고, 다시 이 두 테이블의 데이터를 동시에 출력하거나 관계가 있는 테이블을 참조하기 위해서는 데이터를 연결해야 하는데 이 과정을 조인이라고 함

계좌

계좌번호	예수금	관리점 코드	관리점
100111	100	1000	서울점
100222	200	1001	경기점
100333	300	1002	인천점
100444	400	1003	제주점



계좌

계좌번호	예수금	관리점 코드
100111	100	1000
100222	200	1001
100333	300	1002
100444	400	1003

관리점

관리점 코드	관리점
1000	서울점
1001	경기점
1002	인천점
1003	제주점

☞ 계좌 테이블은 제 3 정규화에 의해 계좌 + 관리점으로 분리됨. 이 때 관리점 코드를 같이 공유함.

만약 계좌 정보와 함께 관리점 정보를 함께 출력(ex. 관리점 별로 거래 계좌의 수 확인)할 경우 두 데이터를 조인하여 출력함. 즉, 두 테이블의 데이터를 함께 출력하거나 참조하기 위해 두 데이터를 연결하는 과정을 조인, 연결키를 조인키라고 함

예제) 계좌번호 100111의 관리점이 어딘지를 찾으려면?

계좌

계좌번호	예수금	관리점 코드	관리점
100111	100	1000	서울점
100222	200	1001	경기점
100333	300	1002	인천점
100444	400	1003	제주점



계좌

계좌번호	예수금	관리점 코드
100111	100	1000
100222	200	1001
100333	300	1002
100444	400	1003

관리점

관리점 코드	관리점
1000	서울점
1001	경기점
1002	인천점
1003	제주점

- ① 계좌번호 테이블에서 계좌번호가 100111 데이터 확인
- ② 계좌번호 테이블에서 계좌번호가 100111 데이터의 관리점 코드(1000)를 확인
- ③ 관리점 코드(1000)를 관리점 테이블에 전달하여 관리점 확인(서울점)

SQL 작성)

```
SELECT A.계좌번호, B.관리점
FROM 계좌 A, 관리점 B
WHERE A.관리점코드 = B.관리점코드
AND A.계좌번호 = '100111';
```

### ● 계층형 데이터 모델

- 자기 자신끼리 관계가 발생. 즉, 하나의 엔터티 내의 인스턴스끼리 계층 구조를 가지는 경우를 말함
- 계층 구조를 갖는 인스턴스끼리 연결하는 조인을 셀프조인이라함(같은 테이블을 여러 번 조인)

예제) 아래 EMP 테이블은 직원테이블로 순서대로 사번, 이름, 직무, 매니저번호, 입사일, 급여, 보너스, 부서번호 컬럼으로 구성. 이 때, 매니저번호(MGR)는 매니저의 사원번호를 의미하므로 사원번호(EMPNO) 컬럼과 관련이 있다. 즉, SMITH의 매니저 이름을 확인하는 과정을 보면, SMITH의 매니저번호를 확인(7902), 해당 번호의 사번을 갖는 데이터를 찾으면 FORD라는 것을 확인할 수 있음

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980/12/17 00:00:00	800		20
2	7499	ALLEN	SALESMAN	7698	1981/02/20 00:00:00	1600	300	30
3	7521	WARD	SALESMAN	7698	1981/02/22 00:00:00	1250	500	30
4	7566	JONES	MANAGER	7839	1981/04/02 00:00:00	2975		20
5	7654	MARTIN	SALESMAN	7698	1981/09/28 00:00:00	1250	1400	30
6	7698	BLAKE	MANAGER	7839	1981/05/01 00:00:00	2850		30
7	7782	CLARK	MANAGER	7839	1981/06/09 00:00:00	2450		10
8	7788	SCOTT	ANALYST	7566	1987/04/19 00:00:00	3000		20
9	7839	KING	PRESI...		1981/11/17 00:00:00	5000		10
10	7844	TURNER	SALESMAN	7698	1981/09/08 00:00:00	1500	0	30
11	7876	ADAMS	CLERK	7788	1987/05/23 00:00:00	1100		20
12	7900	JAMES	CLERK	7698	1981/12/03 00:00:00	950		30
13	7902	FORD	ANALYST	7566	1981/12/03 00:00:00	3000		20
14	7934	MILLER	CLERK	7782	1982/01/23 00:00:00	1300		10

위 예제를 SQL로 표현하면 다음과 같다.

SQL1 \*

```
1 SELECT E1.ENAME AS 사원이름,
2        E2.ENAME AS 매니저이름
3     FROM EMP E1, EMP E2
4    WHERE E1.MGR = E2.EMPNO;
```

Result

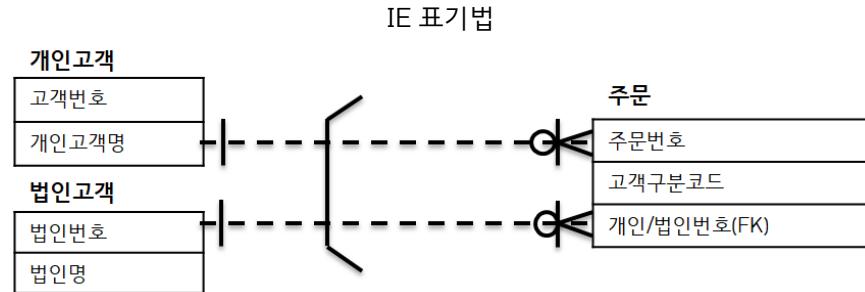
사원이름	매니저이름
1 FORD	JONES
2 SCOTT	JONES
3 TURNER	BLAKE

이하 생략

### ● 상호배타적 관계

- 두 테이블 중 하나만 가능한 관계를 말함

ex) 주문 엔터티에는 개인 또는 법인번호 둘 중 하나만 상속될 수 있음 => 상호배타적 관계  
즉, 주문은 개인고객이거나 법인고객 둘 중 하나의 고객만이 가능



## 1-8. 모델이 표현하는 트랜잭션의 이해

### ● 트랜잭션이란

- 하나의 연속적인 업무 단위를 말함
- 트랜잭션에 의한 관계는 필수적인 관계 형태를 가짐
- 하나의 트랜잭션에는 여러 SELECT, INSERT, DELETE, UPDATE 등이 포함될 수 있음

\* 계좌이체를 예를 들어 A 고객이 B 고객에게 100 만원을 이체하려고 한다고 가정하자.

STEP1) A 고객의 잔액이 100 만원 이상인지 확인

STEP2) 이상이면, A 고객 잔액을 -100 UPDATE

STEP3) B 고객 잔액에 +100 UPDATE

이 때, 2 번과 3 번 과정이 동시에 수행되어야 한다. 즉, 모두 성공하거나 모두 취소되어야 함(All or Nothing)

☞ 이런 특성을 갖는 연속적인 업무 단위를 트랜잭션이라고 한다.

\* 주의

1. A 고객 잔액 차감과 B 고객 잔액 가산이 서로 독립적으로 발생하면 안됨  
-> 각각의 INSERT 문으로 개발되면 안됨

### 2. 부분 COMMIT 불가

-> 동시 COMMIT 또는 ROLLBACK 처리

### ● 필수적, 선택적 관계와 ERD

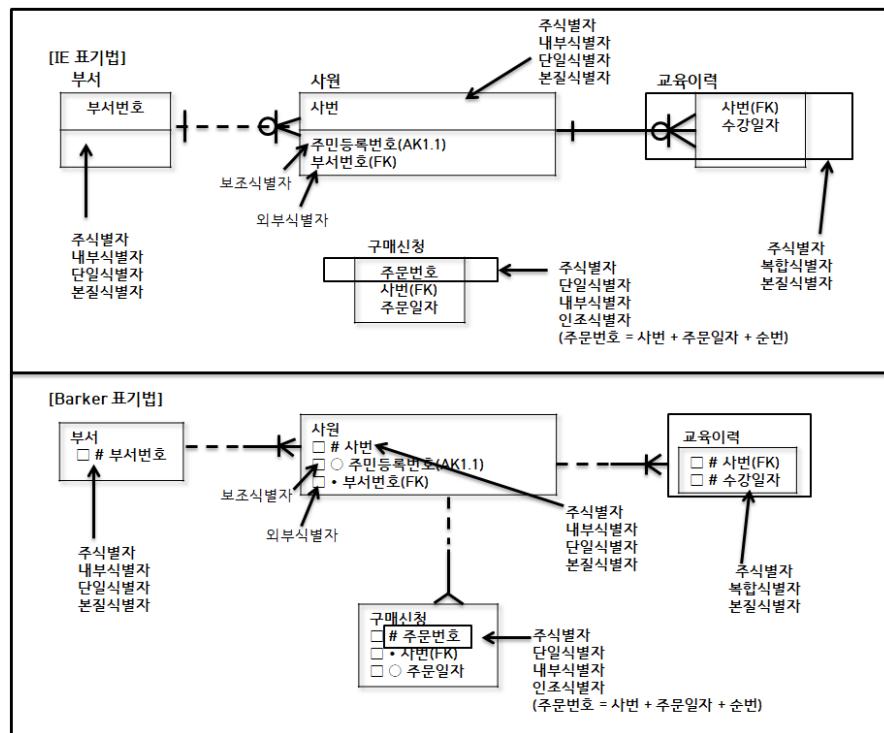
- 두 엔터티의 관계가 서로 필수적일 때 하나의 트랜잭션을 형성
- 두 엔터티가 서로 독립적 수행이 가능하다면 선택적 관계로 표현

## IE 표기법)

- 원을 사용하여 필수적 관계와 선택적 관계를 구분
- 필수적 관계에는 원을 그리지 않는다.
- 선택적 관계에는 관계선 끝에 원을 그린다.

## 바커 표기법)

- 실선과 점선으로 구분
- 필수적 관계는 관계선을 실선으로 표기
- 선택적 관계는 관계선을 점선으로 표기



&lt; 식별자의 분류 - 데이터 모델 &gt;

## 1-9. NULL 속성의 이해

## ● NULL 이란

- DBMS에서 아직 정해지지 않은 값을 의미
- 0과 빈문자열('')과는 다른 개념
- 모델 설계 시 각 컬럼별로 NULL을 허용할지를 결정(Nullable Column)

## ● NULL의 특성

### 1. NULL을 포함한 연산 결과는 항상 NULL

SQL1 \*

```
1 SELECT ENAME, SAL, COMM, SAL + COMM
2 FROM EMP;
```

Result

	ENAME	SAL	COMM	SAL+COMM
1	SMITH	800		
2	ALLEN	1600	300	1900
3	WARD	1250	500	1750
4	JONES	2975		
5	MARTIN	1250	1400	2650
6	BLAKE	2850		
7	CLARK	2450		
8	SCOTT	3000		
9	KING	5000		
10	TURNER	1500	0	1500
11	ADAMS	1100		
12	JAMES	950		
13	FORD	3000		
14	MILLER	1300		

☞ COMM 컬럼에 공백으로 보이는 것들이 NULL이다. (물론 빈 문자열일 수 있지만 해당 데이터에서는 NULL임)

이 때, NULL을 포함한 COMM과 SAL과의 연산결과는 NULL이 리턴된다. (NULL을 사전에 치환한 후 연산 필수)

### ※ NULL 치환 후 연산 결과

SQL1 \*

```
1 SELECT ENAME, SAL, COMM, SAL + NVL(COMM, 0)
2 FROM EMP;
```

Result

	ENAME	SAL	COMM	SAL+NVL(COMM,0)
1	SMITH	800		800
2	ALLEN	1600	300	1900
3	WARD	1250	500	1750
4	JONES	2975		2975
5	MARTIN	1250	1400	2650
6	BLAKE	2850		2850

이하 생략

### 2. 그룹 함수는 NULL을 제외하고 연산을 수행한다.

- SUM, AVG, MIN, MAX 등의 그룹 함수는 항상 NULL을 무시하고 연산

- COUNT는 일반적으로 NULL이 아닌 값의 수만 세지만, COUNT(\*)는 NULL과 상관없이 모든 행의 수를 리턴

### 예제) NULL 을 포함한 컬럼의 집계함수 결과 1

SQL1 *				
1	2	SELECT COUNT(*), COUNT(EMPNO), COUNT(SAL), COUNT(COMM)	FROM EMP;	
Result				
		Grid Result	Server Output	Text Output Explain Plan Statistics
COUNT(*)	COUNT(EMPNO)	COUNT(SAL)	COUNT(COMM)	
1	14	14	14	4

☞ COUNT 는 행의 수를 세는 함수인데, 일반적으로 NULL 은 세지 않으며, NULL 이 아닌 값을 가진 행의 수를 센다. 하지만 COUNT(\*)는 NULL 의 여부와 상관없이 항상 모든 행의 수를 리턴한다.

### 예제) NULL 을 포함한 컬럼의 집계함수 결과 2

SQL1 *				
1	2	SELECT SUM(COMM), MIN(COMM), MAX(COMM)	FROM EMP;	
Result				
		Grid Result	Server Output	Text Output Explain Plan Statistics
SUM(COMM)	MIN(COMM)	MAX(COMM)		
1	2200	0	1400	

☞ SUM, MIN, MAX 연산 결과도 모두 NULL 을 무시하여 연산한다.

### 예제) NULL 을 포함한 컬럼 평균연산

SQL1 *				
1	2	SELECT AVG(COMM), SUM(COMM)/COUNT(*)	FROM EMP;	
Result				
		Grid Result	Server Output	Text Output Explain Plan Statistics
AVG(COMM)	SUM(COMM)/COUNT(*)			
1	550	157.142857142857142857142857142857142857142857		

☞ AVG 연산 결과는 NULL 을 무시한 평균을 리턴하므로, NULL 아닌 4 개의 데이터들의 평균을 리턴. 두 번째 수식은 평균을 직접 구한것으로, COMM 의 총합을 총 행이 수인 14 로 나눈값이다. 따라서 이 두 연산결과는 COMM 이 NULL 을 포함할 경우 다르게 리턴된다. NULL 을 무시한 평균을 얻고자 함인지, 전체 14 명에 대한 평균을 계산하고자 함인지에 따라 적절히 선택하여 사용!

### ● NULL 의 ERD 표기법

- IE 표기법에서는 NULL 허용여부를 알 수 없음
- 바커 표기법에서는 속성 앞에 동그라미가 NULL 허용 속성을 의미함

[IE 표기법]	
주문	
주문번호	
주문금액	
주문최소금액	

[바커 표기법]

주문
<input type="checkbox"/> # 주문번호
<input type="checkbox"/> ○ 주문금액
<input type="checkbox"/> ○ 주문최소금액

## 1-10. 본질식별자와 인조식별자

### ● 식별자 구분(대체 여부에 따른)

#### 1) 본질식별자

- 업무에 의해 만들어지는 식별자(꼭 필요한 식별자)

#### 2) 인조식별자

- 인위적으로 만들어지는 식별자(꼭 필요하지 않지만 관리의 편이성 등의 이유로 인위적으로 만들어지는 식별자)
  - 본질식별자가 복잡한構성을 가질 때 인위적으로 생성
  - 주로 각 행을 구분하기 위한 기본키로 사용되며 자동으로 증가하는 일련번호 같은 형태임

예제) 주문과 주문상세에 대한 엔터티 설계 과정을 예를 들어보자.

주문이 들어오면 주문 엔터티에는 (주문번호 + 고객번호)를 저장, 이 때 PK는 주문번호이다.

주문상세에는 각 주문별로 어떤 상품이, 언제, 몇 개 주문됐는지 등을 기록한다.

주문	주문이력
주문번호	주문번호(FK)
고객번호	상품번호
	주문수량
	주문일자
	배송지

\* 주문상세 테이블 설계 시 다음과 같은 식별자를 고려할 수 있다.

#### 1. PK : 주문번호 + 상품번호로 설계

- 주문을 하면 주문번호와 상품번호가 필요하므로 본질식별자(주문번호 + 상품번호)가 된다
- 하지만 PK가 주문번호 + 상품번호이면 하나의 주문번호로 같은 상품의 주문 결과를 저장할 수 없게 된다.

주문이력
주문번호(FK)
상품번호
주문수량
주문일자
배송지

☞ 실제로 쇼핑을 하다 보면 동일한 장바구니에 A 상품을 5 개 주문했는데, 뒤에 또 다시 A 상품을 3 개 추가로 주문하기도 함

## 2. PK : 주문번호 + 주문순번(주문순번이라는 컬럼을 생성)

- 하나의 주문에 여러 상품에 대한 주문 결과 저장 가능 -> 주문순번으로 인해 구분함

주문이력
주문번호(FK)
주문순번
상품번호
상품명
주문일자
배송지

주문번호	주문순번	상품번호	상품명	주문일자	배송지
0000001	1	100	사과	2024-01-01	서울시
0000001	2	100	사과	2024-01-01	서울시
0000001	3	100	사과	2024-01-01	제주도

☞ 매 주문마다 동일한 상품 주문 시 주문순번을 정하기 위해 상품의 주문 횟수를 세야 한다는 점이 매우 불편!  
즉, 사과를 총 3 번 구매 하였으니 주문순번은 1, 2, 3 순서대로 입력돼야 함

## 3. PK : 주문상세번호(인조식별자 생성)

- 주문상세번호로 각 주문이력을 구분하기 때문에 같은 주문의 같은 상품이력이 저장될 수 있음
- 주문상세번호만이 조식별자이므로 나머지 정보들이 불필요하게 중복 저장될 위험 발생
- 실제 업무와 상관없는 주문상세번호를 조식별자로 생성하면 쓸모없는 INDEX 가 생성됨(PK 생성 시 자동 unique INDEX 생성)

주문이력
주문상세번호
주문번호(FK)
상품번호
상품명
주문일자
배송지

주문상세번호	주문번호	상품번호	상품명	주문일자	배송지
1	0000001	100	사과	2024-01-01	서울시
2	0000001	100	사과	2024-01-01	서울시
3	0000001	100	사과	2024-01-01	제주도

\* 따라서 인조식별자는 다음의 단점을 가지게 된다.

1. **중복 데이터 발생** 가능성 -> 데이터 품질 저하
2. **불필요한 인덱스** 생성 -> 저장공간 낭비 및 DML 성능 저하

\*\* 인덱스는 원래 조회 성능을 향상시키기 위한 객체이며, 인덱스는 DML(INSERT/UPDATE/DELETE)시 INDEX SPLIT 현상으로 인해 성능이 저하된다.

# 2과목.SQL 기본 및 활용

## 2 과목 시험과목 변동사항

추가 :Top N 쿼리, 정규 표현식, PIVOT 절과 UNPIVOT 절

제외 :절차형 SQL(PL/SQL)과 SQL 최적화 기본 원리(인덱스와 조인원리)

### - 2과목 SQL 기본 및 활용(SQLP, SQLD 공통)

변경 전		변경 후	
주요항목	세부항목	주요항목	세부항목
SQL 기본	<ul style="list-style-type: none"> <li>• 관계형 데이터베이스 개요</li> <li>• DDL</li> <li>• DML</li> <li>• TCL</li> <li>• WHERE 절</li> <li>• FUNCTION</li> <li>• GROUP BY, HAVING 절</li> <li>• ORDER BY 절</li> <li>• 조인</li> </ul>	SQL 기본	<ul style="list-style-type: none"> <li>• 관계형 데이터베이스 개요</li> <li>• SELECT 문</li> <li>• 함수</li> <li>• WHERE 절</li> <li>• GROUP BY, HAVING 절</li> <li>• ORDER BY 절</li> <li>• 조인</li> <li>• 표준 조인</li> </ul>
SQL 활용	<ul style="list-style-type: none"> <li>• 표준조인</li> <li>• 집합연산자</li> <li>• 계층형 질의</li> <li>• 서브쿼리</li> <li>• 그룹 함수</li> <li>• 윈도우 함수</li> <li>• DCL</li> <li>• 절차형 SQL</li> </ul>	SQL 활용	<ul style="list-style-type: none"> <li>• 서브 쿼리</li> <li>• 집합 연산자</li> <li>• 그룹 함수</li> <li>• 윈도우 함수</li> <li>• Top N 쿼리</li> <li>• 계층형 질의와 셀프 조인</li> <li>• PIVOT 절과 UNPIVOT 절</li> <li>• 정규 표현식</li> </ul>
SQL 최적화 기본 원리	<ul style="list-style-type: none"> <li>• 옵티マイ저와 실행계획</li> <li>• 인덱스 기본</li> <li>• 조인 수행 원리</li> </ul>	관리 구문	<ul style="list-style-type: none"> <li>• DML</li> <li>• TCL</li> <li>• DDL</li> <li>• DCL</li> </ul>

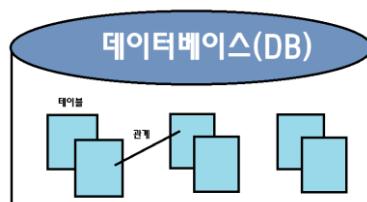
출처 : 한국데이터산업진흥원

## 2-1. 관계형 데이터베이스 개요

### ● 데이터베이스(Database)와 DBMS(Database Management System)

- 데이터베이스 : 데이터의 집합 형식을 갖추지 않아도 엑셀 파일을 모아 둔다면 그것 또한 데이터베이스임
- DBMS : 데이터를 효과적으로 관리하기 위한 시스템

개인이 파일을 여러 개 끊어서 폴더에 보관하면 데이터를 찾고 관리하는데 많은 비용이 발생 이를 보다 시스템적으로 작동하게 만든 시스템을 DBMS라고 한다. (ORACLE, MYSQL 등)



## ● 관계형 데이터베이스 구성 요소

- 계정 : 데이터의 접근 제한을 위한 여러 업무별/시스템별 계정이 존재
- 테이블 : DBMS 의 DB 안에서 데이터가 저장되는 형식
- 스키마 : 테이블이 어떠한 구성으로 되어있는지, 어떠한 정보를 가지고 있는지에 대한 기본적인 구조를 정의

## ● 테이블

### 1. 정의

- 엑셀에서의 워크시트처럼 행(로우)과 열(컬럼)을 갖는 2 차원 구조로 구성, 데이터를 입력하여 저장하는 최소 단위
- 컬럼은 속성이라고도 부름(모델링 단계마다 부르는 용어가 다름)

### 2. 특징

- 하나의 테이블은 반드시 하나의 유저(계정) 소유여야 함
- 테이블간 관계는 일대일(1:1), 일대다(1:N), 다대다(N:N)의 관계를 가질 수 있음
- 테이블명은 중복될 수 없지만, 소유자가 다른 경우 같은 이름으로 생성 가능

ex) SCOTT 소유의 EMP 테이블 존재, HR 소유의 EMP 테이블 생성 가능(같은 계정 내 동일한 객체명 생성 불가)

```
SQL1 *
1 SELECT *
2 FROM DBA_OBJECTS
3 WHERE OBJECT_NAME = 'EMP';
```

OWNER	OBJECT_NAME	SUBOBJECT_NAME	OBJECT_ID	DATA_OBJECT_ID	OBJECT_TYPE
1 SYSTEM	EMP		79986	79986	TABLE
2 SCOTT	EMP		73196	73196	TABLE
3 SCOTT	EMP		79987	79987	INDEX

<생성된 객체 현황>

- 테이블은 행 단위로 데이터가 입력, 삭제되며 수정은 값의 단위로 가능

ex) 사원테이블에 새로운 사원 정보(사원번호, 사원이름 등의)를 테이블 내 모든 컬럼의 값을 동시에 전달하여

입력, 삭제 시에는 해당 사원의 모든 정보가 입력, 삭제됨. 수정 시에는 특정 직원의 급여만 수정 가능

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	
7369	SMITH	CLERK	7902	1980/12/17 00:00:00	800		20	
7499	ALLEN	SALESMAN	7698	1981/02/20 00:00:00	1600	300	30	행 단위 입력/삭제
7521	WARD	SALESMAN	7698	1981/02/22 00:00:00	1250	500	30	
7566	JONES	MANAGER	7839	1981/04/02 00:00:00	2975		20	
7654	MARTIN	SALESMAN	7698	1981/09/28 00:00:00	1250	1400	30	
7698	BLAKE	MANAGER	7839	1981/05/01 00:00:00	2850	특정 값 수정 가능	30	
7782	CLARK	MANAGER	7839	1981/06/09 00:00:00	2450		10	
7788	SCOTT	ANALYST	7566	1987/04/19 00:00:00	3000		20	
7839	KING	PRESIDENT		1981/11/17 00:00:00	5000		10	
7844	TURNER	SALESMAN	7698	1981/09/08 00:00:00	1500	0	30	
7876	ADAMS	CLERK	7788	1987/05/23 00:00:00	1100		20	
7900	JAMES	CLERK	7698	1981/12/03 00:00:00	950		30	
7902	FORD	ANALYST	7566	1981/12/03 00:00:00	3000		20	
7934	MILLER	CLERK	7782	1982/01/23 00:00:00	1300		10	

&lt; 테이블 구조 &gt;

\* 객체 : DBMS에서의 객체는 생성하고 변경할 수 있는 하나의 관리 대상

### ● SQL(Structured Query Language)

- 관계형 데이터베이스에서 데이터 조회 및 조작, DBMS 시스템 관리 기능을 명령하는 언어
- 데이터 정의(DDL), 데이터 조작(DML), 데이터 제어 언어(DCL) 등으로 구분
- SQL 문법은 대소문자를 구분하지 X

### ● 관계형 데이터베이스 특징

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

### ● 데이터 무결성(integrity)

- 데이터의 정확성과 일관성을 유지하고, 데이터에 결손과 부정합이 없음을 보증하는 것
- 데이터베이스에 저장된 값과 그것이 표현하는 현실의 비즈니스 모델의 값이 일치하는 정확성을 의미함
- 데이터 무결성을 유지하는 것이 데이터베이스 관리시스템에 중요한 기능

### ● 데이터 무결성 종류

- ① **개체 무결성** : 테이블의 기본키를 구성하는 컬럼(속성)은 NULL 값이나 중복값을 가질 수 없음
- ② **참조 무결성** : 외래키 값은 NULL 이거나 참조 테이블의 기본키 값과 동일해야 한다.  
(외래키란 참조 테이블의 기본키에 정의된 데이터만 허용되는 구조이므로)
- ③ **도메인 무결성** : 주어진 속성 값이 정의된 도메인에 속한 값이어야 함
- ④ **NULL 무결성** : 특정 속성에 대해 NULL을 허용하지 않는 특징
- ⑤ **고유 무결성** : 특정 속성에 대해, 값이 중복되지 않는 특징

- ⑥ 키 무결성 : 하나의 릴레이션(관계)에는 적어도 하나의 키가 존재해야 함  
(테이블이 서로 관계를 가질 경우 반드시 하나 이상의 조인키를 가짐)

- \* 도메인 : 각 컬럼(속성)이 갖는 범위
- \* 릴레이션 : 테이블간 관계를 말함
- \* 튜플 : 하나의 행을 의미함
- \* 키 : 식별자

### ● ERD(Entity Relationship Diagram)

- ERD란 테이블 간 서로의 상관 관계를 그림으로 표현한 것
- ERD의 구성요소에는 **엔터티(Entity)**, **관계(Relationship)**, **속성(Attribute)**가 있다.  
-> 현실 세계의 데이터는 이 3 가지의 구성으로 모두 표현 가능

## 2-2. SELECT 문

### ● SQL 종류

- SQL은 그 기능에 따라 다음과 같이 구분함

구분	종류
DDL (Data Definition Language)	CREATE, ALTER, DROP, TRUNCATE
DML (Data Manipulation Language)	INSERT, DELETE, UPDATE, MERGE
DCL (Data Control Language)	GRANT, REVOKE
TCL (Transaction Control Language)	COMMIT, ROLLBACK
DQL (Data Query Language)	SELECT

\* 사실 SELECT 문은 따로 SQL 종류 중 어디에도 속하지 않아서 SELECT 문을 위한 DQL 등장

### ● SELECT 문 구조

- SELECT 문은 다음과 같이 **6개 절로 구성**
- 각 절의 순서대로 작성해야 함(GROUP BY 절과 HAVING 절의 순서는 서로 바꿀 수 있지만 보통 사용하지 않음)
- SELECT 문의 내부 파싱(문법적 해석) 순서는 나열된 순서와는 다름
- **FROM > WHERE > GROUP BY > HAVING > SELECT > ORDER BY** 순서대로 실행됨

## \*\* 문법

```
SQL1 *
1 SELECT * | 컬럼명 | 표현식
2 FROM 테이블명 또는 뷰명
3 WHERE 조회 조건
4 GROUP BY 그룹핑컬럼명
5 HAVING 그룹핑 필터링 조건
6 ORDER BY 정렬컬럼명;
```

## ● SELECT 절

- SELECT 문장을 사용하여 불러올 컬럼명, 연산 결과를 작성하는 절
- \*를 사용하여 테이블 내 전체 컬럼명을 불러올 수 있음
- 원하는 컬럼을 콤마(.)로 나열하여 작성 가능(순서대로 출력됨)
- 표현식이란 원래의 컬럼명을 제외한 모든 표현 가능한 가능한 대상(연산식, 기존 컬럼의 함수 변형식 포함)

## \*\* 문법

```
SQL1 *
1 SELECT * | 컬럼명 | 표현식
2 FROM 테이블명 또는 뷰명;
```

## \*\* 특징

- SELECT 절에서 표시할 대상 컬럼에 Alias(별칭) 지정 가능
- 대소문자를 구분하지 않아도 인식한다.

## 예제) EMP 테이블의 전체 컬럼 조회

SQL1 \*

```
1 SELECT *
2 FROM EMP;
```

Result

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980/12/17 00:00:00	800		20
2	7499	ALLEN	SALESMAN	7698	1981/02/20 00:00:00	1600	300	30
3	7521	WARD	SALESMAN	7698	1981/02/22 00:00:00	1250	500	30
4	7566	JONES	MANAGER	7839	1981/04/02 00:00:00	2975		20
5	7654	MARTIN	SALESMAN	7698	1981/09/28 00:00:00	1250	1400	30
6	7698	BLAKE	MANAGER	7839	1981/05/01 00:00:00	2850		30

이하 생략

## 예제) EMP 테이블에서 특정 컬럼 조회

SQL1 \*

```
1 SELECT EMPNO, ENAME, SAL
2 FROM EMP;
```

Result

	EMPNO	ENAME	SAL
1	7369	SMITH	800
2	7499	ALLEN	1600
3	7521	WARD	1250
4	7566	JONES	2975
5	7654	MARTIN	1250
6	7698	BLAKE	2850

이하 생략

### 예제) 표현식을 사용하여 원본과 다른 데이터 출력 가능

SQL1 \*

```
1 ▶ SELECT EMPNO, ENAME, SAL * 1.1
2   FROM EMP;
```

Result

EMPNO	ENAME	SAL*1.1
1	SMITH	880
2	ALLEN	1760
3	WARD	1375
4	JONES	3272.5
5	MARTIN	1375
6	BLAKE	3135

이하 생략

- ☞ SAL \* 1.1이라는 컬럼은 없지만 기존 컬럼의 값을 사용하여 연산결과를 SELECT 절에서 정의하여 출력할 수 있음. 이런 표현 가능한 모든 수식을 표현식이라고 함(함수식, 연산식 등)

### ● 컬럼 Alias(별칭)

- 컬럼명 대신 **출력할 임시 이름 지정**(SELECT 절에서만 정의 가능, 원본 컬럼명은 변경되지 X)
- **컬럼명 뒤에 AS 와 함께 컬럼 별칭 전달(AS는 생략 가능)**

### ● 특징 및 주의사항

- SELECT 문보다 늦게 수행되는 **ORDER BY 절에서만 컬럼 별칭 사용 가능**(그 외 절에서 사용시 에러 발생)
- 한글 사용 가능(한글 지원 캐릭터셋 설정 시)
- **이미 존재하는 예약어는 별칭으로 사용 불가**
  - ex) AVG, COUNT, DECODE, SELECT, FROM 등
- 다음의 경우 별칭에 반드시 쌍따옴표 전달 필요
  - 1) 별칭에 **공백**을 포함하는 경우
  - 2) 별칭에 **특수문자**를 포함하는 경우(" " 제외)
  - 3) 별칭 그대로 전달할 경우(입력한 대소를 그대로 출력하고자 할 때)

### 예제) 별칭 사용 예(AS 생략 가능)

SQL1 \*

```
1 ▶ SELECT EMPNO AS 사번,
2       ENAME  AS 사원이름,
3       SAL   * 1.1 AS NEW_SAL
4   FROM EMP;
```

Result

	사번	사원이름	NEW_SAL
1	7369	SMITH	880
2	7499	ALLEN	1760
3	7521	WARD	1375
4	7566	JONES	3272.5
5	7654	MARTIN	1375
6	7698	BLAKE	3135
7	7782	CLARK	2695
8	7788	SCOTT	3300
9	7839	KING	5500
10	7844	TURNER	1650
11	7876	ADAMS	1210
12	7900	JAMES	1045
13	7902	FORD	3300
14	7934	MILLER	1430

### 예제) 별칭 선언 시 쌍따옴표가 필요한 경우

SQL1 \*

```
1 SELECT EMPNO, ENAME, SAL * 1.1 AS NEW SAL
2 FROM EMP;
```

Result

ORA-00923: FROM 키워드가 필요한 위치에 없습니다.

☞ 쌍따옴표를 사용하지 않아 에러 발생함

SQL1 \*

```
1 SELECT EMPNO, ENAME, SAL * 1.1 AS "NEW SAL"
2 FROM EMP;
```

Result

	EMPNO	ENAME	NEW SAL
1	7369	SMITH	880
2	7499	ALLEN	1760
3	7521	WARD	1375
4	7566	JONES	3272.5

이하 생략

☞ 별칭에 공백 포함 시 반드시 쌍따옴표와 함께 전달

### ● FROM 절

- 데이터를 불러올 테이블명 또는 뷰명 전달
- 테이블 여러 개 전달 가능(쉼마로 구분) -> 조인 조건 없이 테이블명만 나열 시 카티시안 곱 발생 주의!
- **테이블 별칭 선언 가능(ORACLE은 AS 사용 불가, SQL Server는 사용/생략가능)**
  - \* 테이블 별칭 선언 시 컬럼 구분자는 테이블 별칭으로만 전달(테이블명으로 사용 시 에러 발생)
- **ORACLE에서는 FROM 절 생략 불가**(의미상 필요 없는 경우 DUAL 테이블 선언)
  - \* ORACLE 23c 버전부터는 생략 가능
- **SQL Server에서는 FROM 절 필요 없을 경우 생략 가능**(오늘 날짜 조회 시)

\* 뷰 : 테이블과 동일하게 데이터를 조회할 수 있는 객체이지만 테이블처럼 실제 데이터가 저장된 것이 아닌, SELECT 문 결과에 이름을 붙여 그 이름만으로 조회가 가능하도록 한 기능

### 예제) ORACLE에서의 FROM 절 생략 시 에러 발생 케이스(DUAL 테이블 사용)

SQL1 \*

```
1 SELECT 24 * 123
```

Result

ORA-00923: FROM 키워드가 필요한 위치에 없습니다.

☞ ORACLE에서 FROM 절 생략 시 에러 발생

```
SQL1 *
1 SELECT 24 * 123
2   FROM DUAL;
```

Result

	24*123
1	2952

☞ 의미상 FROM 절이 필요 없는 경우 DUAL 전달

#### 예제) 테이블 별칭 사용 예제

```
SQL1 *
1 SELECT E.ENAME, EMP.SAL, E.DEPTNO
2   FROM EMP E
3 WHERE E.DEPTNO = 10;
```

Result

	Grid Result	Server Output	Text Output	Explain Plan	Statistics
ORA-00904: "EMP"."SAL": 부적합한 식별자					

☞ 잘못된 사용 예

```
SQL1 *
1 SELECT E.ENAME, E.SAL, E.DEPTNO
2   FROM EMP E
3 WHERE E.DEPTNO = 10;
```

Result

	ENAME	SAL	DEPTNO
1	CLARK	2450	10
2	KING	5000	10
3	MILLER	1300	10

☞ 테이블 별칭을 선언한 경우 컬럼참조(동일한 이름의 컬럼을 구분하기 위해 테이블명 또는 별칭을 컬럼명 앞에 전달)는 테이블명으로 사용 불가

## 2-3. 함수

### ● 함수 정의

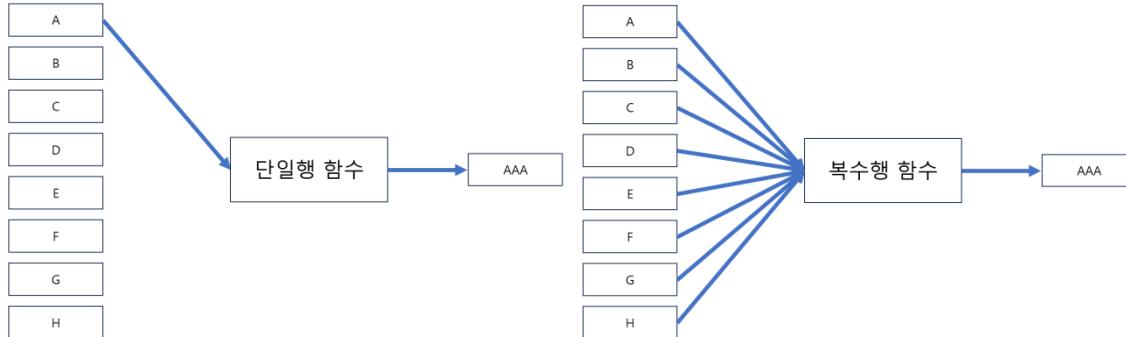
- input value 가 있을 경우 그에 맞는 output value 를 출력해주는 객체
- input value 와 output value 의 관계를 정의한 객체
- from 절을 제외한 모든 절에서 사용 가능

### ● 함수 기능

- 기본적인 쿼리문을 더욱 강력하게 해줌
- 데이터의 계산을 수행
- 개별 데이터의 항목을 수정
- 표시할 날짜 및 숫자 형식을 지정
- 열 데이터의 유형(data type)을 변환

## ● 함수의 종류(입력값의 수에 따라)

- 단일행 함수와 복수행 함수로 구분
- **단일행 함수** : input 과 output 의 관계가 1:1
- **복수행 함수** : 여러 건의 데이터를 동시에 입력 받아서 하나의 요약값을 리턴  
(그룹함수 또는 집계함수라고도 함)



< 단일행 함수와 복수행 함수의 input 과 output 의 관계 >

## ● 입/출력값의 타입에 따른 함수 분류

### 1) 문자형 함수

- 문자열 결합, 추출, 삭제 등을 수행
- 단일행 함수 형태
- output 은 대부분 문자값(LENGTH, INSTR 제외)

### ※ 문자함수 종류

함수명	함수기능	사용예시	출력	기타설명
LOWER(대상)	문자열을 소문자로	LOWER('ABC')	abc	
UPPER(대상)	문자열을 대문자로	UPPER('abc')	ABC	
SUBSTR(대상,m,n)	문자열 중 m위치에서 n개의 문자열 추출	SUBSTR('ABCDE',2,3) SUBSTR('ABCDE',2) SUBSTR('ABCDE',-4,3)	BCD BCDE BCD	n 생략 시 끝까지 추출 뒤에서 4번째(B)부터 오른쪽으로 스캔하여 3개의 문자열 추출
INSTR(대상, 찾을문자열,m,n)	대상에서 찾을문자열 위치 반환 (m위치에서 시작, n번째 발견된 문자열위치)	INSTR('A#B#C#','#') INSTR('A#B#C#','#',3,2) INSTR('A#B#C#','#',-3,2)	2 6 2	m과 n 생략 시 1로 해석 3번째부터 두번째 발견된 # 위치 뒤에서 3번째( # )에서 왼쪽으로 스캔하여 두 번째로 발견된 #의 위치 리턴
LTRIM(대상, 삭제문자열)	문자열 중 특정 문자열을 왼쪽에서 삭제	LTRIM('AABABA','A')	BABA	삭제문자열 생략 시 공백 삭제
RTRIM(대상, 삭제문자열)	문자열 중 특정 문자열을 오른쪽에서 삭제	RTRIM('AABABA','A')	AABA	삭제문자열 생략 시 공백 삭제
TRIM(대상)	문자열 중 특정 문자열을 양쪽에서 삭제	TRIM(' ABCDE ')	ABCDE	ORACLE TRIM은 공백만 삭제 가능
LPAD(대상,n,문자열)	대상 왼쪽에 문자열을 추가하여 총 n의 길이 리턴	LPAD('ABC',5,'**')	**ABC	
RPAD(대상,n,문자열)	대상 오른쪽에 문자열을 추가하여 총 n의 길이 리턴	RPAD('ABC',5,'**')	ABC**	
CONCAT(대상1, 대상2)	문자열 결합	CONCAT('A','B')	AB	두 개의 인수만 전달 가능
LENGTH(대상)	문자열 길이	LENGTH('ABCDE')	5	
REPLACE(대상,찾을문자열,바꿀문자열)	문자열 치환 및 삭제	REPLACE('ABBA','AB','ab')	abBA	세번째 인수를 생략하거나 빈문자열 전달 시 찾을문자열 삭제 가능
TRANSLATE(대상, 찾을문자열,바꿀문자열)	글자를 1대1로 치환	TRANSLATE('ABBA','AB','ab')	abba	매칭되는 글자끼리 치환(A는a로, B는b로 각각) 바꿀문자열 생략 불가, 빈문자열 전달 시 null 리턴

### SQL Server)

- SUBSTR -> SUBSTRING
- LENGTH -> LEN
- INSTR -> CHARINDEX

## 2) 숫자형 함수

- 숫자를 입력하면 숫자 값을 반환
- 단일행 함수 형태의 숫자함수
- ORACLE 과 SQL Server 함수 거의 동일

### \* 숫자함수 종류

함수명	함수기능	사용예시	출력	기타설명
ABS(숫자)	절대값 반환	ABS(-1.5)	1.5	
ROUND(숫자, 자리수)	소수점 특정 자리에서 반올림	ROUND(123.456, 2)	123.46	소수점 둘째자리로 반올림
		ROUND(123.456, -2)	100	자리수가 음수이면 정수자리에서 반올림 (십의 자리에서 반올림 진행)
TRUNC(숫자, 자리수)	소수점 특정 자리에서 버림	TRUNC(123.456, 2)	123.45	
SIGN(숫자)	숫자가 양수면 1 음수면 -1 0이면 0 반환	SIGN(100)	1	
FLOOR(숫자)	작거나 같은 최대 정수 리턴	FLOOR(3.5)	3	
CEIL(숫자)	크거나 같은 최소 정수 리턴	CEIL(3.5)	4	
MOD(숫자1, 숫자2)	숫자1을 숫자2로 나누어 나머지 반환	MOD(7,2)	1	
POWER(m,n)	m의 n 거듭제곱	POWER(2,4)	16	
SQRT(숫자)	루트값 리턴	SQRT(16)	4	

## 3) 날짜형 함수

- 날짜 연산과 관련된 함수
- ORACLE 과 SQL Server 함수 거의 다름

### \* 날짜함수 종류

함수명	함수기능	사용예시	출력	기타설명
SYSDATE	현재 날짜와 시간 리턴	SYSDATE	2024/02/14 18:44:34	날짜 출력 형식에 따라 다르게 출력됨 (날짜만 출력될 수 있음)
CURRENT_DATE	현재 날짜 리턴	CURRENT_DATE	2024/02/14	날짜 출력 형식에 따라 다르게 출력됨 (시간이 출력될 수 있음)
CURRENT_TIMESTAMP	현재 타임스탬프 리턴	CURRENT_TIMESTAMP	2024/02/14 18:45:29 +09:00	
ADD_MONTHS(날짜, n)	날짜에서 n개월 후 날짜 리턴	ADD_MONTHS(SYSDATE, 3)	2024/05/14 18:44:34	n이 음수인 경우 n개월 이전 날짜 리턴
MONTHS_BETWEEN(날짜1, 날짜2)	날짜1과 날짜2의 개월 수 리턴	MONTHS_BETWEEN(SYSDATE, HIREDATE)	3.7234	날짜1 < 날짜2로 전달 시 음수 리턴
LAST_DAY(날짜)	주어진 월의 마지막 날짜 리턴	LAST_DAY(SYSDATE)	2024/02/29 18:44:34	
NEXT_DAY(날짜, n)	주어진 날짜 이후 지정된 요일의 첫 번째 날짜 리턴	NEXT_DAY(SYSDATE, 1)	2024/02/18 18:51:35	1:일요일, 2:월요일, ..., 7:토요일
ROUND(날짜, 자리수)	날짜 반올림	ROUND(SYSDATE, 'MONTH')	2024-02-01 0:00	월 이전자리에서 반올림
TRUNC(날짜, 자리수)	날짜 버림	TRUNC(SYSDATE, 'MONTH')	2024-02-01 0:00	월 이전자리에서 버림

## SQL Server)

- SYSDATE -> GETDATE
- ADD\_MONTHS -> DATEADD(월 뿐만 아니라 모든 단위 날짜 연산 가능)
- MONTHS\_BETWEEN -> DATEDIFF(두 날짜 사이의 년, 월, 일 추출)

## 4) 변환함수

- 값의 **데이터 타입을 변환**
- 문자를 숫자로, 숫자를 문자로, 날짜를 문자로 변경

### \* 변환함수 종류

함수명	함수기능	사용예시	출력	기타설명
TO_NUMBER(문자)	숫자 타입으로 변경하여 리턴	TO_NUMBER('100')	100	문자100을 숫자100으로 리턴
TO_CHAR(대상, 포맷)	1) 날짜의 포맷 변경	TO_CHAR(SYSDATE, 'MM/DD-YYYY')	02/14-2024	날짜 형식 변경(리턴은 문자타입)
	2) 숫자의 포맷 변경	TO_CHAR(9000, '9,999') TO_CHAR(9000, '09999')	9,000 09000	천단위 구분기호 생성(리턴은 문자타입) 총 5자리로 리턴(앞 자리수 0으로)
TO_DATE(문자, 포맷)	주어진 문자를 포맷 형식에 맞게 읽어 날짜로 리턴	TO_DATE('2024/01/01', 'YYYY/MM/DD')	2024/01/01 00:00:00	날짜로 리턴됨
FORMAT(날짜, 포맷)	날짜의 포맷 변경	FORMAT(GETDATE(), 'YYYY')	2024	SQL SERVER 함수
CAST(대상 AS 데이터타입)	대상을 주어진 데이터타입으로 변환	CAST('100' AS int)	100	문자100을 숫자100으로 리턴

### SQL Server)

- TO\_NUMBER, TO\_DATE, TO\_CHAR -> CONVERT(포맷 전달 시)
- 단순 변환일 경우 주로 CAST 사용

### 5) 그룹함수

- 다중행 함수
- 여러 값이 input값으로 들어가서 하나의 요약된 값으로 리턴
- GROUP BY와 함께 자주 사용됨
- ORACLE과 SQL Server 거의 동일

### \* 그룹함수 종류

함수명	함수기능	사용예시	출력	기타설명
COUNT(대상)	행의 수 리턴	SELECT COUNT(SAL) FROM EMP;	각 연산 결과	NULL 무시하고 연산
SUM(대상)	총 합 리턴	SELECT SUM(SAL) FROM EMP;	각 연산 결과	NULL 무시하고 연산
AVG(대상)	평균 리턴	SELECT AVG(SAL) FROM EMP;	각 연산 결과	NULL 무시하고 연산
MIN(대상)	최솟값 리턴	SELECT MIN(SAL) FROM EMP;	각 연산 결과	NULL 무시하고 연산
MAX(대상)	최댓값 리턴	SELECT MAX(SAL) FROM EMP;	각 연산 결과	NULL 무시하고 연산
VARIANCE(대상)	분산 리턴	SELECT VARIANCE(SAL) FROM EMP;	각 연산 결과	NULL 무시하고 연산
STDDEV(대상)	표준편차 리턴	SELECT STDDEV(SAL) FROM EMP;	각 연산 결과	NULL 무시하고 연산

### SQL Server)

- VARIANCE -> VAR
- STDDEV -> STDEV

### 6) 일반함수

- 기타 함수(널 치환 함수 등)

### \* 일반(기타)함수 종류

함수명	함수기능	사용예시	출력	기타설명
DECODE(대상, 값1, 리턴1, 값2, 리턴2, ..., 그외리턴)	대상이 값1이면 리턴1, 값2와 같으면 리턴2, .. 그외에는 그외리턴값 리턴	DECODE(DEPTNO, 10, A, B)	A 또는 B	대소비교에 따른 치환 불가 그외리턴 생략 시 널 리턴
NVL(대상, 치환값)	대상이 널이면 치환값으로 치환하여 리턴	NVL(COMM, 0)	COMM값 또는 0리턴	
NVL2(대상, 치환값1, 치환값2)	대상이 널이면 치환값2로 치환, 널이 아니면 치환값1로 치환하여 리턴	NVL2(COMM, COMM*1.1, 0)	COMM*1.1값 또는 0리턴	COMM값이 널이면 0, 널이 아니면 COMM*1.1 리턴
COALESCE(대상1, 대상2, ..., 그외리턴)	대상을 중 널이면 아닌값 출력(가장 첫번째부터) 대상3, ... 모두가 널이면 그외리턴값이 리턴됨	COALESCE(NULL, 100)	100	그외리턴값 생략 시 널 리턴
ISNULL(대상, 치환값)	대상이 널이면 치환값이 리턴	ISNULL(NULL, 100)	100	SQL SERVER 함수
NULLIF(대상1, 대상2)	두 값이 같으면 널 리턴, 다르면 대상1 리턴	NULLIF(10, 20)	10	
CASE문	조건별 치환 및 연산 수행	밀에 참고	밀에 참고	

### 예제) DECODE 사용1

```
SQL1 *
1 SELECT DEPTNO,
2      DECODE(DEPTNO, 10, '인사부', 20, '재무부') AS DNAME
3   FROM EMP;
```

Result

DEPTNO	DNAME
1	20 재무부
2	30
3	30
4	20 재무부
5	30
6	30
7	10 인사부
8	20 재무부

이하 생략

☞ 부서번호가 10번이면 인사부, 20번이면 재무부, 나머지는 널 리턴

### 예제) DECODE 사용2

```
SQL1 *
1 SELECT DEPTNO, JOB,
2      DECODE(DEPTNO,
3              10,
4              DECODE(JOB, 'CLERK', 'A', 'B'), 'C') AS RESULT
5   FROM EMP;
```

Result

DEPTNO	JOB	RESULT
1	10 CLERK	A
2	10 MANAGER	B
3	10 PRESIDENT	B
4	20 ANALYST	C
5	20 ANALYST	C
6	20 CLERK	C
7	20 CLERK	C
8	20 MANAGER	C

이하 생략

☞ DEPTNO가 10이면서 JOB이 CLERK이면 A, DEPTNO가 10이면서 CLERK가 아니면 B, DEPTNO가 10이 아니면 C

### 예제) NVL, NVL2 사용

```
SQL1 *
1 SELECT COMM,
2      NVL(COMM, 0) AS RESULT1,
3      NVL2(COMM, COMM * 1.1, 500) AS RESULT2
4   FROM EMP;
```

Result

COMM	RESULT1	RESULT2
	0	500
300	300	330
500	500	550
	0	500
1400	1400	1540
	0	500
	0	500

이하 생략

☞ NVL2의 경우 NVL이랑 다르게 COMM의 값이 널이 아닐 때도 치환값 정의 가능  
 ☞ NVL2의 경우 COMM이 널이 아니면 10% 인상값, 널이면 500 리턴

### 예제) COALESCE 사용

```
SQL1 *
1 SELECT DEPTNO1,
2      DEPTNO2,
3      COALESCE(DEPTNO1, DEPTNO2, 0) AS RESULT
4  FROM STUDENT;
```

Result

Grid Result Server Output Text Output Explain Plan Statistics

	DEPTNO1	DEPTNO2	RESULT
1	101	201	101
2	102		102
3	103	203	103
4	201		201

이하 생략

☞ DEPTNO1과 DEPTNO2중 널이 아닌 값 출력(둘 다 널이 아니면 맨 앞 순서대로 출력).

모두 널이면 0 출력

### 예제) CASE문 사용1

```
SQL1 *
1 SELECT SAL,
2      CASE WHEN SAL < 2000 THEN 'C'
3            WHEN SAL < 3000 THEN 'B'
4            ELSE 'A'
5      END AS SAL_GRADE
6  FROM EMP;
```

Result

Grid Result Server Output Text Output Explain Plan Statistics

	SAL	SAL_GRADE
1	800	C
2	1600	C
3	1250	C
4	2975	B
5	1250	C
6	2850	B
7	2450	B
8	3000	A

이하 생략

☞ CASE문을 사용하여 여러 조건(대소비교 가능)에 대한 치환 및 연산 가능!

### 예제) CASE문 사용2

SQL1 \*

```

1  SELECT DEPTNO,
2      CASE DEPTNO WHEN 10 THEN '인사부'
3          WHEN 20 THEN '총무부'
4          WHEN 30 THEN '재무부'
5          ELSE '기타'
6      END AS DNAME1,
7      CASE WHEN DEPTNO = 10 THEN '인사부'
8          WHEN DEPTNO = 20 THEN '총무부'
9          WHEN DEPTNO = 30 THEN '재무부'
10         ELSE '기타'
11     END AS DNAME2
12
13  FROM EMP;

```

Result

	DEPTNO	DNAME1	DNAME2
1	10	인사부	인사부
2	10	인사부	인사부
3	10	인사부	인사부
4	20	총무부	총무부
5	20	총무부	총무부
6	20	총무부	총무부
7	20	총무부	총무부
8	20	총무부	총무부
9	30	재무부	재무부
10	30	재무부	재무부

이하 생략

- ☞ 동등비교 시 위처럼 비교대상(DEPTNO)를 CASE와 WHEN 사이에 배치하면서 WHEN절마다 반복하지 않아도 됨 (이 때 DEPTNO 데이터 타입과 WHEN절의 명시된 비교대상의 데이터타입 반드시 일치해야 함)

### 예제) NULLIF 사용

SQL1 \*

```

1  SELECT ENAME, SAL, COMM,
2      NULLIF(COMM, 100) AS 결과1,
3      NULLIF(COMM, 500) AS 결과2
4
5  FROM EMP
6  WHERE DEPTNO = 30;

```

Result

	ENAME	SAL	COMM	결과1	결과2
1	ALLEN	1600	300	300	300
2	WARD	1250	500	500	
3	MARTIN	1250	1400	1400	1400
4	BLAKE	2850			
5	TURNER	1500	0	0	0
6	JAMES	950			

- ☞ NULLIF 는 널을 갖는 값을 치환하기 위한 목적이 아닌, 특정 값과 일치하는 대상을 NULL로 치환하기 위해 사용함. 따라서 결과 1의 경우 COMM의 값이 100과 일치하는 행이 없으므로 원래 값을 유지, 결과 2의 경우 COMM의 값이 500인 WARD의 COMM만 NULL로 변경되어 출력됨

### 예제) ISNULL 사용(SQL SERVER 실행)

SQL1 \*

```

1  SELECT ENAME, SAL, COMM,
2      ISNULL(COMM, 100) AS 결과
3  FROM EMP
4  WHERE DEPTNO = 30;

```

Result

	ENAME	SAL	COMM	결과
1	ALLEN	1600	300	300
2	WARD	1250	500	500
3	MARTIN	1250	1400	1400
4	BLAKE	2850	100	100
5	TURNER	1500	0	0
6	JAMES	950	100	100

## 2-4. WHERE 절

### ● WHERE 절

- 테이블의 데이터 중 원하는 조건에 맞는 데이터만 조회하고 싶을 경우 사용  
(엑셀의 필터기능과 유사)
- 여러 조건 동시 전달 가능(AND 와 OR 로 조건 연결)
- **NULL 조회 시 IS NULL / IS NOT NULL 연산자 사용(= 연산자로 조회 불가)**
- 연산자를 사용하여 다양한 표현이 가능
- 조건 전달 시 비교 대상의 데이터 타입 일치하는 것이 좋음

ex) EMP 테이블의 부서번호 컬럼의 데이터타입은 숫자인데 문자상수로 비교 시 성능 문제 발생할 수 있음

연산자 종류	설명
=	같은 조건을 검색
!=, <>	같지 않은 조건을 검색
>	큰 조건을 검색
>=	크거나 같은 조건을 검색
<	작은 조건을 검색
<=	작거나 같은 조건을 검색
BETWEEN a AND b	A 와 B사이에 있는 범위 값을 모두 검색
IN(a, b, c)	A 이거나 B 이거나 C 인 조건을 검색
LIKE	특정 패턴을 가지고 있는 조건을 검색
Is Null / Is Not Null	Null 값을 검색 / Null 이 아닌 값을 검색
A AND B	A 조건과 B 조건을 모두 만족하는 값만 검색
A OR B	A 조건이나 B조건 중 한가지라도 만족하는 값을 검색
NOT A	A 가 아닌 모든 조건을 검색

### \*\* 문법

SQL1 \*

```

1  SELECT * | 컬럼명 | 표현식
2  FROM 테이블명 또는 뷰명
3  WHERE 조회할 데이터 조건;

```

\*\* 주의사항

- 문자나 날짜 상수 표현 시 반드시 훌따옴표 사용(다른 절에서도 동일 적용)
- ORACLE 은 문자 상수의 경우 대소문자를 구분
- SQL Server 은 기본적으로 문자상수의 대소문자를 구분하지 X

예제) 이름이 SMITH 인 직원 조회(조건절에 문자 상수 사용)

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL
2   FROM EMP
3 WHERE ENAME = 'SMITH';
```

Result

EMPNO	ENAME	SAL
1	SMITH	800

☞ ORACLE 의 경우 테이블에 데이터는 실제로 대문자로 저장되어 있으면 소문자로 조회 시 데이터 출력 안됨

예제) SAL 이 1500 이상인 사원 정보 조회(숫자 상수 전달)

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL
2   FROM EMP
3 WHERE SAL >= 1500;
```

Result

EMPNO	ENAME	SAL
1	ALLEN	1600
2	JONES	2975
3	BLAKE	2850
4	CLARK	2450

이하 생략

예제) COMM 값이 NULL 인 직원 정보 출력

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL, COMM
2   FROM EMP
3 WHERE COMM IS NULL;
```

Result

EMPNO	ENAME	SAL	COMM
1	SMITH	800	
2	JONES	2975	
3	BLAKE	2850	
4	CLARK	2450	
5	SCOTT	3000	

이하 생략

예제) 여러 조건 전달 - AND 연산자의 사용 (DEPTNO 가 10번이면서 SAL 이 2000 이상인 직원 출력)

```
SQL1 *
1 SELECT EMPNO, ENAME, SAL, COMM, DEPTNO
2   FROM EMP
3 WHERE DEPTNO = 10
4   AND SAL >= 2000;
```

Result

	EMPNO	ENAME	SAL	COMM	DEPTNO
1	7782	CLARK	2450		10
2	7839	KING	5000		10

☞ 두 조건이 모두 만족하는 대상을 찾을 경우 AND 연산자 사용

예제) 여러 조건 전달 - OR 연산자의 사용 (DEPTNO 가 10번 이거나 SAL 이 2000 이상인 직원 정보 출력)

```
SQL1 *
1 SELECT EMPNO, ENAME, SAL, COMM, DEPTNO
2   FROM EMP
3 WHERE DEPTNO = 10
4   OR SAL >= 2000;
```

Result

	EMPNO	ENAME	SAL	COMM	DEPTNO
1	7566	JONES	2975		20
2	7698	BLAKE	2850		30
3	7782	CLARK	2450		10
4	7788	SCOTT	3000		20
5	7839	KING	5000		10

이하 생략

☞ 위와 똑같은 조건이지만 두 조건을 연결하는 논리연산자를 AND(모두 만족)로 사용할 때와 결과가 다름.  
OR은 두 조건 중 하나만 성립해도 되는, 두 조건 결과의 합집합을 출력하는 논리연산자임

### ● IN 연산자

- 포함연산자로 여러 상수와 일치하는 조건 전달 시 사용
- 상수를 괄호로 묶어서 동시에 전달(문자와 날짜 상수의 경우 반드시 흑따옴표와 함께)

예제) SMITH 와 SCOTT 의 직원 정보 출력

```
SQL1 *
1 SELECT EMPNO, ENAME, SAL, COMM, DEPTNO
2   FROM EMP
3 WHERE ENAME = 'SMITH'
4   OR ENAME = 'SCOTT';
```

Result

	EMPNO	ENAME	SAL	COMM	DEPTNO
1	7369	SMITH	800		20
2	7788	SCOTT	3000		20

☞ 이름이 SMITH 이면서 SCOTT 일 수는 없으므로 두 조건을 각각 만족하는 합집합을 구하라는 의미임.  
하지만 동일한 조건대상(ENAME)이 계속 반복돼야 하는 불편함이 있음 -> IN 연산자 사용

## \*\* IN 연산자로 변경

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL, COMM, DEPTNO
2   FROM EMP
3 WHERE ENAME IN ('SMITH', 'SCOTT');
```

Result

	EMPNO	ENAME	SAL	COMM	DEPTNO
1	7369	SMITH	800		20
2	7788	SCOTT	3000		20

- ☞ IN 연산자를 사용하면 조건대상(ENAME)과 연산자(=)의 반복을 줄일 수 있음.  
이 때, ()안의 상수도 문자상수와 날짜상수는 훌따옴표 필수

## ● BETWEEN A AND B 연산자

- A 보다 크거나 같고 B 보다 작거나 같은 조건을 만족
- A 와 B 에는 범위로 묶을 상수값 전달(문자, 숫자, 날짜 모두 전달 가능)
- 반드시 A 가 B 보다 작아야 함(반대로 작성 시 아무것도 출력되지 X)

### 예제) SAL 이 2000 이상 3000 이하인 직원 정보 출력

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL, COMM, DEPTNO
2   FROM EMP
3 WHERE SAL >= 2000
4   AND SAL <= 3000;
```

Result

	EMPNO	ENAME	SAL	COMM	DEPTNO
1	7566	JONES	2975		20
2	7698	BLAKE	2850		30
3	7782	CLARK	2450		10
4	7788	SCOTT	3000		20
5	7902	FORD	3000		20

- ☞ 역시 SAL 이 반복되는 특징을 보임 => BETWEEN A AND B 연산자 사용

## \*\* BETWEEN 연산자 사용

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL, COMM, DEPTNO
2   FROM EMP
3 WHERE SAL BETWEEN 2000 AND 3000;
```

Result

	EMPNO	ENAME	SAL	COMM	DEPTNO
1	7566	JONES	2975		20
2	7698	BLAKE	2850		30
3	7782	CLARK	2450		10
4	7788	SCOTT	3000		20
5	7902	FORD	3000		20

- ☞ BETWEEN 연산자를 사용하면 SAL 에 대한 반복을 할 필요 없음

### 예제) BETWEEN 연산자 주의사항

The screenshot shows the SQL1\* tab with the following SQL code:

```

1 SELECT EMPNO, ENAME, SAL, COMM, DEPTNO
2 FROM EMP
3 WHERE SAL BETWEEN 3000 AND 2000;

```

Below the code is a Result pane with tabs: Grid Result, Server Output, Text Output, Explain Plan, Statistics. The Grid Result tab is selected, showing a table with columns: EMPNO, ENAME, SAL, COMM, DEPTNO. The table is currently empty.

At the bottom left, there is a status bar with the word "Ready".

☞ BEWEEN A AND B 에서 A 가 B 보다 더 큰 값일 경우 아무것도 조회되지 않음

### ● LIKE 연산자

- 정확하게 일치하지 않아도 되는 패턴 조건 전달 시 사용
- %와 \_와 함께 사용됨
  - 1) % : 자리수 제한 없는 모든이라는 의미
  - 2) \_ : \_ 하나 당 한 자리수를 의미하며 모든 값을 표현함

### 예제) LIKE 연산자

- ENAME LIKE 'S%' : 이름이 S로 시작하는
- ENAME LIKE '%S%' : 이름에 S를 포함하는
- ENAME LIKE '%S' : 이름이 S로 끝나는
- ENAME LIKE '\_S%' : 이름의 두 번째 글자가 S인(맨 앞이 \_인것 주의! %이면 자리수 상관없이 S를 포함하기만 하면 됨)
- ENAME LIKE '\_\_S\_\_' : 이름의 가운데 글자가 S이며 이름의 길이가 5글자인

### ● NOT 연산자

- 조건 결과의 반대집합. 즉, 여집합을 출력하는 연산자
- NOT 뒤에 오는 연산 결과의 반대 집합 출력
- 주로 NOT IN, NOT BETWEEN A AND B, NOT LIKE, NOT NULL로 사용

### 예제) NOT 연산자의 사용

The screenshot shows the SQL1\* tab with the following SQL code:

```

1 SELECT EMPNO, ENAME, SAL, COMM, DEPTNO
2 FROM EMP
3 WHERE NOT(SAL > 3000);

```

Below the code is a Result pane with tabs: Grid Result, Server Output, Text Output, Explain Plan, Statistics. The Grid Result tab is selected, showing a table with columns: EMPNO, ENAME, SAL, COMM, DEPTNO. The table contains four rows of data:

	EMPNO	ENAME	SAL	COMM	DEPTNO
1	7369	SMITH	800		20
2	7499	ALLEN	1600	300	30
3	7521	WARD	1250	500	30
4	7566	JONES	2975		20

A callout box labeled "이하 생략" (omitted below) points to the bottom of the table.

☞ 이 경우 "3000 보다 작거나 같은" 조건으로 변경 가능하므로 위처럼 NOT 을 사용하지 않는다.

SQL1 \*

```

1 SELECT EMPNO, ENAME, SAL, COMM, DEPTNO
2   FROM EMP
3 WHERE SAL NOT BETWEEN 1000 AND 3000;

```

Result

	EMPNO	ENAME	SAL	COMM	DEPTNO
1	7369	SMITH	800		20
2	7839	KING	5000		10
3	7900	JAMES	950		30

☞ 1000 이상 3000 이하의 반대 집합 -> 1000 미만 또는 3000 초과\

## 2-5. GROUP BY 절

### ● GROUP BY 절

- 각 행을 특정 조건에 따라 그룹으로 분리하여 계산하도록 하는 구문식
- GROUP BY 절에 그룹을 지정할 컬럼을 전달(여러 개 전달 가능)
- 만약 그룹 연산에서 제외할 대상이 있다면 미리 WHERE 절에서 해당 행을 제외함  
(WHERE 절이 GROUP BY 절보다 먼저 수행되므로)
- 그룹에 대한 조건은 WHERE 절에서 사용할 수 없음
- SELECT 절에 집계 함수를 사용하여 그룹연산 결과 표현
- GROUP BY 절을 사용하면 데이터가 요약되므로 요약되기 전 데이터와 함께 출력할 수 없음

### \*\* 문법

SQL1 \*

```

1 SELECT * | 컬럼명 | 표현식
2   FROM 테이블명 또는 뷔명
3 WHERE 조회할 데이터 조건
4 GROUP BY 그룹핑컬럼명
5 HAVING 그룹핑 대상 필터링 조건;

```

### 예제) 부서별(DEPARTMENT\_ID) 급여 총합과 급여 평균 출력

SQL1 \*

```

1 SELECT DEPARTMENT_ID, SUM(SALARY),
2       ROUND(AVG(SALARY)) AS AVG_SALARY
3   FROM EMPLOYEES
4  GROUP BY DEPARTMENT_ID;

```

Result

	DEPARTMENT_ID	SUM(SALARY)	AVG_SALARY
1	100	51608	8601
2	30	24900	4150
3		7000	7000
4	20	19000	9500
5	70	10000	10000
6	90	58000	19333
7	110	20308	10154

이하 생략

### 예제) 직군(JOB\_ID)별 급여 총합과 급여 평균 출력

SQL1 \*

```

1 SELECT JOB_ID, SUM(SALARY),
2        ROUND(AVG(SALARY)) AS AVG_SALARY
3     FROM EMPLOYEES
4    GROUP BY JOB_ID;

```

Result

JOB_ID	SUM(SALARY)	AVG_SALARY
1 AC_MGR	12008	12008
2 AC_ACCOUNT	8300	8300
3 IT_PROG	28800	5760
4 ST_MAN	36400	7280
5 AD_ASST	4400	4400
6 PU_MAN	11000	11000
7 SH_CLERK	64300	3215

이하 생략

### 예제) GROUP BY 의 잘못된 사용

SQL1 \*

```

1 SELECT DEPTNO, MAX(SAL), ENAME
2   FROM EMP
3  GROUP BY DEPTNO;

```

Result

DEPTNO	MAX(SAL)	ENAME
10	5000	KING
20	3000	ADVISOR
30	3000	CLERK
40	3000	MGR
50	3000	SALESMAN
60	3000	ST_CLERK

☞ GROUP BY 절에 DEPTNO 를 사용하면 DEPTNO 가 같은 값끼리 묶여서 요약 정보만 SELECT 절에 표현 가능. 따라서 GROUP BY 컬럼과 집계 함수를 사용한 결과만이 전달 가능

=> GROUP BY 절에 명시되지 않은 컬럼 전달 불가!

### ● HAVING 절

- 그룹 함수 결과를 조건으로 사용할 때 사용하는 절
- WHERE 절을 사용하여 그룹을 제한할 수 없으므로 HAVING 절에 전달
- HAVING 절이 GROUP BY 절 앞에 올 수는 있지만 뒤에 쓰는 것을 권장
- 내부적 연산 순서가 SELECT 절보다 먼저이므로 SELECT 절에서 선언된 Alias 사용 불가

### 예제) 그룹 함수 조건을 WHERE 절에 전달하는 경우 발생한 에러

SQL1 \*

```

1 SELECT DEPARTMENT_ID,
2        SUM(SALARY)
3     FROM EMPLOYEES
4    WHERE SUM(SALARY) > 20000
5   GROUP BY DEPARTMENT_ID;

```

Result

DEPARTMENT_ID	SUM(SALARY)
10	100000
20	100000
30	100000
40	100000
50	100000
60	100000

ORA-00934: 그룹 함수는 허가되지 않습니다

### 예제) 그룹 함수 조건 HAVING 절에 전달

SQL1 \*

```

1 SELECT DEPARTMENT_ID,
2      SUM(SALARY)
3  FROM EMPLOYEES
4 GROUP BY DEPARTMENT_ID
5 HAVING SUM(SALARY) > 2000;

```

Result

DEPARTMENT_ID	SUM(SALARY)
1	100
2	30
3	7000
4	20
5	70
6	90
7	110

이하 생략

### 예제) WHERE 절과 HAVING 절 동시 사용

SQL1 \*

```

1 > SELECT DEPARTMENT_ID,
2      SUM(SALARY)
3  FROM EMPLOYEES
4 WHERE DEPARTMENT_ID IN (10, 30, 40, 110)
5 HAVING SUM(SALARY) > 20000
6 GROUP BY DEPARTMENT_ID;

```

Result

DEPARTMENT_ID	SUM(SALARY)
1	30
2	110

☞ 순서상 WHERE 절을 먼저 수행, 원하는 데이터만 필터링 한 후 GROUP BY에 의해 그룹연산을 수행한 뒤 HAVING 절에서 만족하는 데이터만 선택하여 출력함

## 2-6. ORDER BY 절

### ● ORDER BY 절

- 데이터는 입력된 순서대로 출력되나, 출력되는 행의 순서를 사용자가 변경하고자 할 때 ORDER BY 절을 사용
- ORDER BY 뒤에 명시된 컬럼 순서대로 정렬 -> 1차 정렬, 2차 정렬 전달 가능
- 정렬 순서를 오름차순(ASC), 내림차순(DESC)으로 전달(생략 시 오름차순 정렬)
- 유일하게 SELECT 절에 정의한 컬럼 별칭 사용 가능
- SELECT 절에 선언된 순서대로의 숫자 전달 가능(컬럼명과 숫자 혼합 사용가능)

### \*\*문법

SQL1 \*

```

1 SELECT * | 컬럼명 | 표현식
2 FROM 테이블명 또는 부여
3 WHERE 조회할 데이터 조건
4 GROUP BY 그룹핑컬럼명
5 HAVING 그룹핑 대상 필터링 조건
6 ORDER BY 정렬컬럼명 [ASC|DESC];

```

## ● 정렬 순서(오름차순)

- 한글 : 가, 나, 다, 라...
- 영어 : A, B, C, D...
- 숫자 : 1, 2, 3, 4...
- 날짜 : 과거 날짜부터 시작해서 최근 날짜로 정렬

### 예제) 문자 정렬

\* 문자는 원쪽부터 값이 작은 순서대로, 같은 값이면 두 번째 값이 작은 순서대로 정렬된다.

```
SQL1 *
1 SELECT FIRST_NAME, EMPLOYEE_ID, DEPARTMENT_ID
2   FROM EMPLOYEES
3  WHERE DEPARTMENT_ID = 30
4 ORDER BY FIRST_NAME;
```

	FIRST_NAME	EMPLOYEE_ID	DEPARTMENT_ID
1	Alexander	115	30
2	Den	114	30
3	Guy	118	30
4	Karen	119	30
5	Shelli	116	30
6	Sigal	117	30

### 예제) 숫자값을 문자값으로 바꾼 뒤 정렬

```
SQL1 *
1 SELECT TO_CHAR(SAL) AS SAL
2   FROM EMP
3  WHERE SAL <= 2000
4 ORDER BY SAL;
```

	SAL
1	1100
2	1250
3	1250
4	1300
5	1500
6	1600
7	950

이하 생략

### 예제) 날짜 정렬(오름차순 : 오래된 순서대로)

SQL1 \*

```

1 SELECT FIRST_NAME, EMPLOYEE_ID,
2      DEPARTMENT_ID, HIRE_DATE
3  FROM EMPLOYEES
4 WHERE DEPARTMENT_ID = 30
5 ORDER BY HIRE_DATE;

```

Result

FIRST_NAME	EMPLOYEE_ID	DEPARTMENT_ID	HIRE_DATE
Den	114	30	2002/12/07 00:00:00
Alexander	115	30	2003/05/18 00:00:00
Sigal	117	30	2005/07/24 00:00:00
Shelli	116	30	2005/12/24 00:00:00
Guy	118	30	2006/11/15 00:00:00
Karen	119	30	2007/08/10 00:00:00

### 예제) SELECT 절 컬럼 순서를 사용한 정렬

SQL1 \*

```

1 SELECT EMPLOYEE_ID, FIRST_NAME, HIRE_DATE
2   FROM EMPLOYEES
3 ORDER BY 3;

```

Result

EMPLOYEE_ID	FIRST_NAME	HIRE_DATE
102	Lex	2001/01/13 00:00:00
203	Susan	2002/06/07 00:00:00
204	Hermann	2002/06/07 00:00:00
205	Shelley	2002/06/07 00:00:00
206	William	2002/06/07 00:00:00
109	Daniel	2002/08/16 00:00:00
108	Nancy	2002/08/17 00:00:00

이하 생략

### ● 복합 정렬

- 먼저 정렬한 값의 동일한 결과가 있을 경우 추가적으로 정렬 가능  
=> 1차 정렬한 값이 같은 경우 그 값 안에서 2차 정렬 컬럼값의 정렬이 일어남

### 예제) SALARY 값을 기준으로 내림차순으로 먼저 정렬 후,

동일한 SALARY 값이 있을 경우 HIRE\_DATE 값으로 한 번 정렬

SQL1 \*

```

1 SELECT FIRST_NAME, SALARY, HIRE_DATE
2   FROM EMPLOYEES
3 WHERE SALARY > 10000
4   AND DEPARTMENT_ID = 90
5 ORDER BY SALARY DESC, HIRE_DATE ASC;

```

Result

FIRST_NAME	SALARY	HIRE_DATE
Steven	24000	2003/06/17 00:00:00
Lex	17000	2001/01/13 00:00:00
Neena	17000	2005/09/21 00:00:00

### 예제) 컬럼 별칭을 사용한 정렬

SQL1 \*

```

1 SELECT EMPLOYEE_ID AS EID,
2       SALARY,
3       DEPARTMENT_ID
4   FROM EMPLOYEES E
5 WHERE DEPARTMENT_ID = 100
6 ORDER BY EID DESC;

```

Result

	EID	SALARY	DEPARTMENT_ID
1	113	6900	100
2	112	7800	100
3	111	7700	100
4	110	8200	100
5	109	9000	100
6	108	12008	100

☞ SELECT 절보다 늦게 수행되는 구문은 ORDER BY 절 뿐이므로 ORDER BY 절만 SELECT 절에서 정의된 컬럼 별칭 사용 가능

### 예제) 컬럼명 또는 컬럼별칭과 컬럼순서를 사용한 정렬

SQL1 \*

```

1 SELECT FIRST_NAME, EMPLOYEE_ID,
2       DEPARTMENT_ID, HIRE_DATE
3   FROM EMPLOYEES
4 WHERE DEPARTMENT_ID IN (10, 20, 30)
5 ORDER BY DEPARTMENT_ID, 2;

```

Result

	FIRST_NAME	EMPLOYEE_ID	DEPARTMENT_ID	HIRE_DATE
1	Jennifer	200	10	2003/09/17 00:00:00
2	Michael	201	20	2004/02/17 00:00:00
3	Pat	202	20	2005/08/17 00:00:00
4	Den	114	30	2002/12/07 00:00:00
5	Alexander	115	30	2003/05/18 00:00:00
6	Shelli	116	30	2005/12/24 00:00:00
7	Sigal	117	30	2005/07/24 00:00:00
8	Guy	118	30	2006/11/15 00:00:00
9	Karen	119	30	2007/08/10 00:00:00

### ● NULL의 정렬

- NULL을 포함한 값의 정렬 시 ORACLE은 기본적으로 NULL을 마지막에 배치(SQL Server는 처음에 배치)
- ORACLE은 ORDER BY 절에 NULLS LAST | NULLS FIRST을 명시하여 NULL 정렬 순서 변경 가능

### 예제) NULL 을 포함한 컬럼의 정렬 결과(ORACLE)

The screenshot shows the SQL1 editor with the following query:

```

1 SELECT ENAME, COMM, SAL
2 FROM EMP
3 WHERE DEPTNO = 30
4 ORDER BY COMM;

```

The Result tab displays the output in a grid format:

	ENAME	COMM	SAL
1	TURNER	0	1500
2	ALLEN	300	1600
3	WARD	500	1250
4	MARTIN	1400	1250
5	JAMES		950
6	BLAKE		2850

☞ NULLS LAST 가 기본이므로 NULL 이 마지막에 배치

The screenshot shows the SQL1 editor with the following query:

```

1 SELECT ENAME, COMM, SAL
2 FROM EMP
3 WHERE DEPTNO = 30
4 ORDER BY COMM NULLS FIRST;

```

The Result tab displays the output in a grid format:

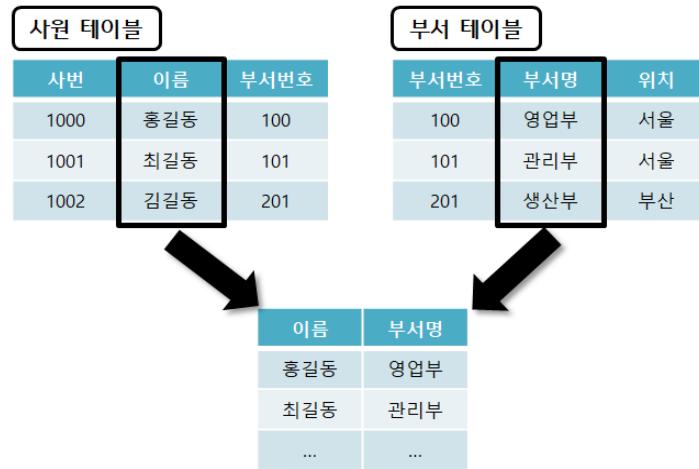
	ENAME	COMM	SAL
1	JAMES		950
2	BLAKE		2850
3	TURNER	0	1500
4	ALLEN	300	1600
5	WARD	500	1250
6	MARTIN	1400	1250

☞ NULLS FIRST로 NULL 정렬 위치 변경 가능

## 2-7. 조인

### ● JOIN(조인)

- 여러 테이블의 데이터를 사용하여 동시 출력하거나 참조 할 경우 사용
- FROM 절에 조인할 테이블 나열
- ORACLE 표준은 테이블 나열 순서 중요하지 X, ANSI 표준은 OUTER JOIN 시 순서 중요
- WHERE 절에서 조인 조건을 작성(ORACLE 표준)
- 동일한 열 이름이 여러 테이블에 존재할 경우 열 이름 앞에 테이블 이름이나 테이블 Alias 붙임
- N 개의 테이블을 조인하려면 최소 N-1 개의 조인 조건이 필요
- ORACLE 표준과 ANSI 표준이 서로 다름



## ● 조인 종류

### 1. 조건의 형태에 따라

1) EQUI JOIN(등가 JOIN) : JOIN 조건이 동등 조건인 경우

2) NON EQUI JOIN: JOIN 조건이 동등 조건이 아닌 경우

### 2. 조인 결과에 따라

1) INNER JOIN : JOIN 조건에 성립하는 데이터만 출력하는 경우

2) OUTER JOIN: JOIN 조건에 성립하지 않는 데이터도 출력하는 경우

(LEFT/RIGHT/FULL OUTER JOIN 으로 나뉨)

3. NATURAL JOIN : 조인조건 생략 시 두 테이블에 같은 이름으로 자연 연결되는 조인

4. CROSS JOIN : 조인조건 생략 시 두 테이블의 발생 가능한 모든 행을 출력하는 조인

5. SELF JOIN: 하나의 테이블을 두 번 이상 참조하여 연결하는 조인

## ● EQUI JOIN(등가 JOIN)

- 조인 조건이 '='(equal) 비교를 통해 같은 값을 가지는 행을 연결하여 결과를 얻는 조인 방법

- SQL 명령문에서 가장 많이 사용하는 조인 방법

- FROM 절에 조인하고자 하는 테이블을 모두 명시

- FROM 절에 명시하는 테이블은 테이블 별칭(Alias) 사용 가능

- WHERE 절에 두 테이블의 공통 컬럼에 대한 조인 조건을 나열

T1	
NO	NAME
1	A
2	B
3	C
NULL	D

T2	
NO	NAME
1	A
1	B
2	C
NULL	D

EQUI JOIN(T1.NO = T2.NO)			
T1.NO	T2.NO	T1.NAME	T2.NAME
1	1	A	A
1	1	A	B
2	2	B	C

## \*\* 문법(ORACLE 표준)

```
SQL1 *
1 SELECT 테이블1.컬럼, 테이블2.컬럼
2   FROM 테이블1, 테이블2
3 WHERE 테이블1.컬럼 = 테이블2.컬럼;
```

예제) EMP 테이블과 DEPT 테이블을 사용하여 각 직원의 이름과 부서명을 함께 출력

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7566	JONES	2975	20
7654	MARTIN	1250	30
7698	BLAKE	2850	30
7782	CLARK	2450	10
7788	SCOTT	3000	20
7839	KING	5000	10
7844	TURNER	1500	30
7876	ADAMS	1100	20
7900	JAMES	950	30
7902	FORD	3000	20
7934	MILLER	1300	10

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

## &lt; 정답 &gt;

SQL1 *	
1	SELECT EMP.ENAME, DEPT.DNAME
2	FROM EMP, DEPT
3	WHERE EMP.DEPTNO = DEPT.DEPTNO;

Result

	ENAME	DNAME
1	CLARK	ACCOUNTING
2	KING	ACCOUNTING
3	MILLER	ACCOUNTING
4	JONES	RESEARCH
5	FORD	RESEARCH
6	ADAMS	RESEARCH
7	SMITH	RESEARCH
8	SCOTT	RESEARCH
9	WARD	SALES
10	TURNER	SALES
11	ALLEN	SALES
12	JAMES	SALES
13	BLAKE	SALES
14	MARTIN	SALES

## ● NON-EQUI JOIN

- 테이블을 연결짓는 조인 컬럼에 대한 비교 조건이 '<', BETWEEN A AND B 와 같이 '=' 조건이 아닌 연산자를 사용하는 경우의 조인조건

\*\*문법

SQL1 *			
1	SELECT	테이블1. 컬럼, 테이블2. 컬럼	
2	FROM	테이블1, 테이블2	
3	WHERE	테이블1. 컬럼 비교조건 테이블2. 컬럼;	

예제) EMP 테이블의 급여를 확인하고 SAL\_GRADE 에 있는 급여 등급 기준에 따라 직원이름과 급여, 급여등급 출력

EMP TABLE

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7566	JONES	2975	20
7654	MARTIN	1250	30
7698	BLAKE	2850	30
7782	CLARK	2450	10
7788	SCOTT	3000	20
7839	KING	5000	10
7844	TURNER	1500	30
7876	ADAMS	1100	20
7900	JAMES	950	30
7902	FORD	3000	20
7934	MILLER	1300	10

SALGRADE TABLE

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

출력 대상

&lt; 정답 &gt;

SQL1 *			
1	▶ SELECT	E.ENAME, E.SAL, S.GRADE	
2	FROM	EMP E, SALGRADE S	
3	WHERE	E.SAL BETWEEN S.LOSAL AND S.HISAL;	

Result

	ENAME	SAL	GRADE
1	SMITH	800	1
2	JAMES	950	1
3	ADAMS	1100	1
4	WARD	1250	2
5	MARTIN	1250	2
6	MILLER	1300	2
7	TURNER	1500	3
8	ALLEN	1600	3

이하 생략

### ● 세 테이블 이상의 조인

- 관계를 잘 파악하여 모든 테이블이 연결되도록 조인 조건 명시
- N 개 테이블의 경우 최소 N-1 개의 조인 조건 필요

## 예제) EMPLOYEES, DEPARTMENTS, LOCATIONS 테이블 조인

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	MANAGER_ID	DEPARTMENT_ID
198	Donald	OConnell	124	50
199	Douglas	Grant	124	50
200	Jennifer	Whalen	101	10
201	Michael	Hartstein	100	20
202	Pat	Fay	201	20
203	Susan	Mavris	101	40
204	Hermann	Baer	101	70
205	Shelley	Higgins	101	110
206	William	Gietz	205	110
100	Steven	King		90
101	Neena	Kochhar	100	90
102	Lex	De Haan	100	90

EMPLOYEES TABLE

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury		1700
130	Corporate Tax		1700

DEPARTMENTS TABLE

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY
1000	1297 Via Cola di Rie	00989	Roma
1100	93091 Calle della Testa	10934	Venice
1200	2017 Shinjuku-ku	1689	Tokyo
1300	9450 Kamiya-cho	6823	Hiroshima
1400	2014 Jabberwocky Rd	26192	Southlake
1500	2011 Interiors Blvd	99236	South San Francisco
1600	2007 Zagora St	50090	South Brunswick
1700	2004 Charade Rd	98199	Seattle
1800	147 Spadina Ave	M5V 2L7	Toronto
1900	6092 Boxwood St	Y5W 9T2	Whitehorse
2000	40-5-12 Laogianggen	190518	Beijing
2100	1298 Vileparle (E)	490231	Bombay

LOCATIONS TABLE

&lt; 정답 &gt;

SQL1 \*

```

1 SELECT E.EMPLOYEE_ID, E.FIRST_NAME, E.LAST_NAME,
2      D.DEPARTMENT_NAME, L.COUNTRY_ID
3   FROM EMPLOYEES E, DEPARTMENTS D, LOCATIONS L
4 WHERE E.DEPARTMENT_ID = D.DEPARTMENT_ID
5   AND D.LOCATION_ID = L.LOCATION_ID;

```

Result

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_NAME	COUNTRY_ID
1	198	Donald	OConnell	Shipping	US
2	199	Douglas	Grant	Shipping	US
3	200	Jennifer	Whalen	Administration	US
4	201	Michael	Hartstein	Marketing	CA
5	202	Pat	Fay	Marketing	CA
6	203	Susan	Mavris	Human Resources	UK
7	204	Hermann	Baer	Public Relations	DE
8	205	Shelley	Higgins	Accounting	US
9	206	William	Gietz	Accounting	US
10	100	Steven	King	Executive	US
11	101	Neena	Kochhar	Executive	US
12	102	Lex	De Haan	Executive	US

이하 생략

☞ 만약 필수 조인조건이 하나라도 생략될 경우 카티시안곱 발생  
(정상 조인보다 더 많은 수의 행이 리턴)

### ● SELF JOIN

- 한 테이블 내 각 행끼리 관계를 갖는 경우 사용하는 조인 기법
- 한 테이블을 참조할 때마다(필요할 때마다) 명시해야 함
- 테이블명이 중복되므로 반드시 테이블 별칭 사용

예제) EMPLOYEES 테이블에서의 각 직원이름과 매니저 이름을 함께 출력

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID
100	Steven	King	24000	
101	Neena	Kochhar	17000	100
102	Lex	De Haan	17000	100
103	Alexander	Hunold	9000	102
104	Bruce	Ernst	6000	103
105	David	Austin	4800	103
106	Valli	Pataballa	4800	103
107	Diana	Lorentz	4200	103
108	Nancy	Greenberg	12008	101
109	Daniel	Faviet	9000	108
110	John	Chen	8200	108
111	Ismael	Sciarra	7700	108
112	Jose Manuel	Urman	7800	108
113	Luis	Popp	6900	108
114	Den	Raphaely	11000	100

&lt; 정답 &gt;

SQL1 \*

```

1 SELECT E1.EMPLOYEE_ID, E1.FIRST_NAME, E1.LAST_NAME, E1.MANAGER_ID,
2       E2.EMPLOYEE_ID, E2.FIRST_NAME, E2.LAST_NAME
3   FROM EMPLOYEES E1, EMPLOYEES E2
4 WHERE E1.MANAGER_ID = E2.EMPLOYEE_ID
5 ORDER BY 1, 4;

```

Result

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	MANAGER_ID	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
1	101	Neena	Kochhar	100	100	Steven	King
2	102	Lex	De Haan	100	100	Steven	King
3	103	Alexander	Hunold	102	102	Lex	De Haan
4	104	Bruce	Ernst	103	103	Alexander	Hunold
5	105	David	Austin	103	103	Alexander	Hunold
6	106	Valli	Pataballa	103	103	Alexander	Hunold
7	107	Diana	Lorentz	103	103	Alexander	Hunold
8	108	Nancy	Greenberg	101	101	Neena	Kochhar
9	109	Daniel	Faviet	108	108	Nancy	Greenberg
10	110	John	Chen	108	108	Nancy	Greenberg
11	111	Ismael	Sciarra	108	108	Nancy	Greenberg
12	112	Jose Manuel	Urman	108	108	Nancy	Greenberg
13	113	Luis	Popp	108	108	Nancy	Greenberg

이하 생략

- ☞ E1 의 MANAGER\_ID 와 E2 의 EMPLOYEE\_ID 가 같은 라인만 출력!
- ☞ EMPLOYEE\_ID 가 100 인 직원은 MANAGER\_ID 가 NULL 이므로 조건에 맞지 않아 생략됨  
(ORACLE 은 INNER JOIN 이 기본 조인 연산임)
- ☞ 조인 조건이 일치하지 않은 데이터를 추가적으로 출력이 되길 원할 경우 OUTER JOIN 수행 필요!

### 응용 예제) EMP 테이블에서 상위관리자(매니저)보다 급여가 많은 직원 출력

중요 1) 테이블 한 번 스캔 시(한 행만 읽었을 경우) 매니저 정보는 없으므로 셀프 조인 필요

중요 2) 원하는 정보를 모두 한 행으로 출력 후 조건 선택 가능

중요 3) 조인조건과 일반조건을 각자의 위치에 전달

(ORACLE 은 모두 WHERE 절 기술, ANSI 표준은 조인조건은 ON 절, 일반조건만 WHERE 절)

SQL1 \*

```

1 SELECT E1.EMPNO, E1.ENAME, E1.SAL,
2       E2.EMPNO, E2.ENAME, E2.SAL
3   FROM EMP E1, EMP E2
4 WHERE E1.MGR = E2.EMPNO
5   AND E1.SAL > E2.SAL;

```

Result

	EMPNO	ENAME	SAL	EMPNO	ENAME	SAL
1	7902	FORD	3000	7566	JONES	2975
2	7788	SCOTT	3000	7566	JONES	2975

## 2-8. 표준 조인

### ● 표준조인

- ANSI 표준으로 작성되는 INNER JOIN, CROSS JOIN, NATURAL JOIN, OUTER JOIN 을 말함

### ● INNER JOIN

- 내부 조인이라고 하며, 조인 조건이 일치하는 행만 추출([ORACLE 조인 기본](#))
- ANSI 표준의 경우 FROM 절에 INNER JOIN 혹은 줄여서 JOIN 을 명시
- ANSI 표준의 경우 [USING이나 ON 조건절을 필수적으로 사용](#)

### ● ON 절

- 조인할 양 컬럼의 컬럼명이 서로 다르더라도 사용 가능
- [ON 조건의 괄호는 옵션\(생략가능\)](#)
- 컬럼명이 같을 경우 테이블 이름이나 별칭을 사용하여 명확하게 지정(테이블 출처 명확히)
- ON 조건절에서 조인조건 명시, WHERE 절에서는 일반조건 명시(WHERE 절과 ON 절을 쓰임에 따라 명확히 구분)

#### \*\* 문법

```
SQL1 *
1 SELECT 테이블1.컬럼명, 테이블2.컬럼명
2 FROM 테이블1 INNER JOIN 테이블2
3   ON 테이블1.조인컬럼 = 테이블2.조인컬럼;
```

예제) EMP 테이블과 DEPT 테이블을 사용하여 각 직원의 이름과 부서명을 함께 출력(EQUI JOIN)

ENAME	DNAME
CLARK	ACCOUNTING
KING	ACCOUNTING
MILLER	ACCOUNTING
JONES	RESEARCH
FORD	RESEARCH
ADAMS	RESEARCH
SMITH	RESEARCH
SCOTT	RESEARCH

이하 생략

☞ ORACLE 표준은 FROM 절에 테이블을 컴마로 구분, WHERE 절에 조인 조건 나열

☞ ORACLE은 INNER JOIN 이 기본 조인 연산이므로 별도의 문법 존재 안함

## ● USING 조건절

- 조인할 컬럼명이 같을 경우 사용
- Alias 나 테이블 이름 같은 접두사 붙이기 불가
- 괄호 필수

\*\*문법

```
SQL1 *
1 SELECT 테이블1.컬럼명, 테이블2.컬럼명
2   FROM 테이블1 INNER JOIN 테이블2
3   USING (동일컬럼명);
```

예제) USING 절을 이용한 사원이름과 부서이름 조회

SQL1 \*

```
1 ▶ SELECT EMP.ENAME, DEPT.DNAME
2   FROM EMP JOIN DEPT
3   USING (DEPTNO);
```

Result

	ENAME	DNAME
1	CLARK	ACCOUNTING
2	KING	ACCOUNTING
3	MILLER	ACCOUNTING
4	JONES	RESEARCH
5	FORD	RESEARCH
6	ADAMS	RESEARCH
7	SMITH	RESEARCH
8	SCOTT	RESEARCH

이하 생략

## ● NATURAL JOIN

- 두 테이블 간의 동일한 이름을 가지는 모든 컬럼들에 대해 EQUI JOIN 을 수행
- USING, ON, WHERE 절에서 조건 정의 불가
- JOIN 에 사용된 컬럼들은 데이터 유형이 동일해야 하며 접두사를 사용불가

T1	
NO	NAME
1	A
2	B
NULL	C

T2	
NO	NAME
1	A
2	C
NULL	C

NATURAL JOIN			
T1.NO	T2.NO	T1.NAME	T2.NAME
1	1	A	A

\*\*문법

SQL1 \*

```
1 SELECT 테이블1.컬럼명, 테이블2.컬럼명
2   FROM 테이블1 NATURAL JOIN 테이블2;
```

### 예제) NATURAL 조인을 이용한 사원 이름, 부서명 출력

SQL1 \*

```
1 SELECT EMP.ENAME, DEPT.DNAME
2   FROM EMP NATURAL JOIN DEPT;
```

Result

	ENAME	DNAME
1	CLARK	ACCOUNTING
2	KING	ACCOUNTING
3	MILLER	ACCOUNTING

이하 생략

### 예제) NATURAL JOIN 시 주의

SQL1 \*

```
1 SELECT *
2   FROM STUDENT NATURAL JOIN PROFESSOR;
```

Result

NAME	PROFNO	STUDNO	SID	GRADE	JUMIN	BIRTHDAY	TEL	HEIGHT
------	--------	--------	-----	-------	-------	----------	-----	--------

- ☞ NATURAL JOIN 은 동일한 이름의 모든 컬럼을 조인 컬럼으로 사용하므로 조인 컬럼의 값이 모두 같을 때만 결과가 리턴 됨
- ☞ STUDENT 와 PROFESSOR 테이블에는 NAME 컬럼과 PROFNO 컬럼이 컬럼명이 서로 동일함

### ● CROSS JOIN

- 테이블 간 JOIN 조건이 없는 경우 생성 가능한 모든 데이터들의 조합  
(Cartesian product(카타시안곱) 출력)
- 양쪽 테이블 행의 수의 곱한 수의 데이터 조합 발생( $m * n$ )

T1		T2		CROSS JOIN			
NO	NAME	NO	NAME	T1.NO	T2.NO	T1.NAME	T2.NAME
1	A	1	A	1	1	A	A
2	B	2	B	1	2	A	B
NULL	C			2	1	B	A
				2	2	B	B
				NULL	1	C	A
				NULL	2	C	B

### \*\*문법

SQL1 \*

```
1 SELECT 테이블1.컬럼명, 테이블2.컬럼명
2   FROM 테이블1 CROSS JOIN 테이블2;
```

### 예제) CROSS 조인

SQL1 \*

```
1 SELECT EMP.ENAME, DEPT.DNAME
2   FROM EMP CROSS JOIN DEPT;
```

Result

	ENAME	DNAME
1	SMITH	ACCOUNTING
2	ALLEN	ACCOUNTING
3	WARD	ACCOUNTING
4	JONES	ACCOUNTING
5	MARTIN	ACCOUNTING
6	BLAKE	ACCOUNTING
7	CLARK	ACCOUNTING

이하 생략

☞ 내용이 길어 일부만 출력했지만, 총 56 건이 출력됨(EMP 14 건, DEPT 4 건이므로  $14 \times 4 = 56$ )

## ● OUTER JOIN

- INNER JOIN 과 대비되는 조인방식
- JOIN 조건에서 동일한 값이 없는 행도 반환할 때 사용
- 두 테이블 중 한쪽에 NULL 을 가지면 EQUI JOIN 시 출력되지 않음 -> 이를 출력 시 OUTER JOIN 사용
- 테이블 기준 방향에 따라 LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN 으로 구분
- OUTER 생략 가능(LEFT OUTER JOIN -> LEFT JOIN)

### \*\* OUTER JOIN 종류

#### 1) LEFT OUTER JOIN

- FROM 절에 나열된 왼쪽 테이블에 해당하는 데이터를 읽은 후, 우측 테이블에서 JOIN 대상 읽어옴
- 즉, 왼쪽 테이블이 기준이 되어 오른쪽 테이블 데이터를 채우는 방식
- 우측 값에서 같은 값이 없는 경우 NULL 값으로 출력

#### 2) RIGHT OUTER JOIN

- LEFT OUTER JOIN 의 반대
- 즉, 오른쪽 테이블 기준으로 왼쪽 테이블 데이터를 채우는 방식
- FROM 절에 테이블 순서를 변경하면 LEFT OUTER JOIN 으로 수행 가능

#### 3) FULL OUTER JOIN

- 두 테이블 전체 기준으로 결과를 생성하여 중복 데이터는 삭제 후 리턴
- LEFT OUTER JOIN 결과와 RIGHT OUTER JOIN 결과의 UNION 연산 리턴과 동일함
- ORACLE 표준에는 없음

### 예제) LEFT OUTER JOIN

STUDENT 테이블과 PROFESSOR 테이블을 조인하여 1, 4 학년 학생들의 이름, 학년, 지도교수이름 출력

STUDENT TABLE

STUDNO	NAME	GRADE	PROFNO
5001	강성근	4	1001
5002	고대영	4	2001
5003	권정현	4	3002
5004	김건일	4	4001
5005	김재현	4	4003
8001	이유나	1	
8002	이민정	1	
8003	전영훈	1	
8004	허민기	1	
8005	최훈	1	

PROFESSOR TABLE

PROFNO	NAME	POSITION	DEPTNO
1001	강경연	정교수	101
1002	경윤영	조교수	101
1003	김다훈	전임강사	101
2001	노경우	전임강사	102
2002	이용준	조교수	102
2003	장요한	정교수	102
3001	전홍범	정교수	103
3002	정성용	조교수	103
3003	백승윤	전임강사	103
4001	한병두	정교수	201
4002	현기숙	조교수	201
4003	오하영	조교수	202
4004	김만중	전임강사	202

연결  
일치하는 대상 없음

- STUDENT, PROFESSOR 테이블을 PROFNO로 연결하면 학생, 지도교수 정보 함께 출력 가능
- STUDENT 테이블의 PROFNO가 NULL인 경우는 데이터가 생략됨(INNER JOIN 수행)
- 지도교수가 없는 학생 정보 출력 시 OUTER JOIN 수행
- 이 때, 기준이 되는 데이터(생략되지 않았으면 하는 쪽)는 STUDENT 테이블!! => LEFT OUTER JOIN

### ORACLE 표준)

```
SQL1 *
1 SELECT *
2   FROM STUDENT S, PROFESSOR P
3  WHERE S.PROFNO = P.PROFNO(+)
4    AND S.GRADE IN (1,4);
```

- ORACLE 표준은 WHERE 절에 조인 조건을 작성하므로 LEFT OUTER JOIN을 기술하지 X
- WHERE 절에서 기준이 되는 테이블(STUDENT) 반대 테이블 조건 컬럼 뒤에 (+)를 붙임

### ANSI 표준)

```
SQL1 *
1 SELECT S.STUDNO, S.NAME AS 학생명, S.GRADE, S.PROFNO,
2       P.PROFNO, P.NAME AS 교수명
3     FROM STUDENT S LEFT OUTER JOIN PROFESSOR P
4       ON S.PROFNO = P.PROFNO
5      WHERE S.GRADE IN (1,4);
```

Result

STUDNO	학생명	GRADE	PROFNO	PROFNO	교수명
1 5001	강성근	4	1001	1001	강경연
2 5002	고대영	4	2001	2001	노경우
3 5003	권정현	4	3002	3002	정성용
4 5004	김건일	4	4001	4001	한병두
5 5005	김재현	4	4003	4003	오하영
6 8005	최훈	1			INNER JOIN시 출력되지 X
7 8004	허민기	1			LEFT OUTER JOIN시 출력
8 8003	전영훈	1			
9 8002	이민정	1			
10 8001	이유나	1			

- ANSI 표준에서는 조인의 종류를 FROM 절에 테이블과 테이블 사이 명시
- 조인 조건을 바로 뒤에 ON 절에 나열
- WHERE 절은 ON 절 밑에 전달(순서 중요)

### 예제) FULL OUTER JOIN (위 조인 결과를 FULL OUTER JOIN 수행(ANSI 표준으로))

SQL1 \*

```

1 ▶ SELECT S.STUDNO, S.NAME AS 학생명, S.GRADE, S.PROFNO,
2      P.PROFNO, P.NAME AS 교수명
3      FROM STUDENT S FULL OUTER JOIN PROFESSOR P
4        ON S.PROFNO = P.PROFNO;

```

Result

STUDNO	학생명	GRADE	PROFNO	PROFNO	교수명
14	7004 전영훈	2	4003	4003	오하영
15	7005 최슬기	2	4007	4007	하지만
16	8001 이유나	1			
17	8002 이민정	1			
18	8003 전영훈	1			
19	8004 허민기	1			
20	8005 최훈	1			
21			3001	3001	전홍범
22			4006	4006	최지현
23			2003	2003	장요한
24			4005	4005	조유진
25			3003	3003	백승윤
26			1003	1003	김다훈

둘 다 존재

LEFT OUTER JOIN 결과

RIGHT OUTER JOIN 결과

이하 생략

- 7004, 7005 번 학생 정보는 LEFT OUTER JOIN에서도, RIGHT OUTER JOIN에서도 출력됨
- LEFT OUTER JOIN 결과와 RIGHT OUTER JOIN 결과를 동시 출력(중복 데이터는 한 번만)
- ORACLE에서는 지원하지 않는 문법((+)) 기호를 양 방향 전달 시 에러 발생)
- 성능적으로도 좋지 않기 때문에 사용 시 주의 필요!

### ORACLE 문법

SQL1 \*

```

1 ▶ SELECT S.STUDNO, S.NAME AS 학생명, S.GRADE, S.PROFNO,
2      P.PROFNO, P.NAME AS 교수명
3      FROM STUDENT S, PROFESSOR P
4      WHERE S.PROFNO = P.PROFNO(+)
5      UNION
6      SELECT S.STUDNO, S.NAME AS 학생명, S.GRADE, S.PROFNO,
7      P.PROFNO, P.NAME AS 교수명
8      FROM STUDENT S, PROFESSOR P
9      WHERE S.PROFNO(+) = P.PROFNO;

```

Result

STUDNO	학생명	GRADE	PROFNO	PROFNO	교수명
13	7003 홍은희	2	4002	4002	현기숙
14	7004 전영훈	2	4003	4003	오하영
15	7005 최슬기	2	4007	4007	하지만
16	8001 이유나	1			
17	8002 이민정	1			
18	8003 전영훈	1			
19	8004 허민기	1			
20	8005 최훈	1			
21			1003	1003	김다훈
22			2003	2003	장요한

이하 생략

## 2-9. 서브쿼리

### ● 서브쿼리

- 하나의 SQL 문 안에 포함된 또 다른 SQL 문을 의미
- 반드시 괄호로 묶어야 함

ex) SELECT 안에 SELECT 문, INSERT, UPDATE, DELETE 안의 SELECT 문

### ● 서브쿼리 사용 가능한 위치

- ① SELECT 절
- ② FROM 절
- ③ WHERE 절
- ④ HAVING 절
- ⑤ ORDER BY 절
- ⑥ 기타 DML(INSERT, DELETE, UPDATE)절

### \*\* GROUP BY 절 사용 불가

### ● 서브쿼리 종류

#### 1. 동작하는 방식에 따라

##### 1) UN-CORRELATED(비연관) 서브쿼리

- 서브쿼리에 메인쿼리(외부쿼리) 테이블의 컬럼을 포함하지 않은 형태의 서브쿼리
- 서브쿼리가 메인쿼리의 값을 참조하지 않고 독립적으로 실행됨
- 서브쿼리는 한 번만 실행됨

##### 2) CORRELATED(연관) 서브쿼리

- 서브쿼리에 메인쿼리(외부쿼리) 테이블의 컬럼을 포함하는 형태의 서브쿼리
- 서브쿼리가 메인쿼리의 컬럼을 참조하고 있기 때문에 서브쿼리의 실행이 메인쿼리와 독립적이지 않음
- **메인 쿼리의 각 행에 대해 서브쿼리가 실행됨**

#### 예제) 연관 서브쿼리와 비연관 서브쿼리 비교

```
SQL1 *
1 > SELECT ENAME, SAL
2   FROM EMP E1
3 WHERE E1.SAL > (SELECT AVG(E2.SAL)
4                   FROM EMP E2);
```

Result

	ENAME	SAL
1	JONES	2975
2	BLAKE	2850
3	CLARK	2450
4	SCOTT	3000
5	KING	5000
6	FORD	3000

☞ SAL이라는 컬럼이 동일하게 사용된 것처럼 보이지만 메인쿼리는 E1 테이블의 SAL 을, 서브쿼리는 E2 테이블의 SAL 을 각각 사용하기 때문에 서로 독립적으로 수행됨  
서브쿼리 먼저 실행되어 SAL의 평균값을 구한뒤 해당 값을 메인쿼리에 전달하면서 데이터를 선택함

```

SQL1 *
1 SELECT ENAME, SAL
2   FROM EMP E1
3 WHERE E1.SAL > (SELECT AVG(E2.SAL)
4                   FROM EMP E2
5                  WHERE E2.DEPTNO = E1.DEPTNO);

```

Result

	ENAME	SAL
1	ALLEN	1600
2	JONES	2975
3	BLAKE	2850
4	SCOTT	3000
5	KING	5000
6	FORD	3000

☞ 서브쿼리에서 메인쿼리의 컬럼인 E1.DEPTNO 를 참조하고 있기 때문에 서로 독립적으로 수행되지 않고  
메인쿼리의 각 행마다 서브쿼리가 실행됨

## 2. 위치에 따라

### 1) 스칼라 서브쿼리

- SELECT 절에 사용하는 서브쿼리
- 서브쿼리 결과를 마치 **하나의 컬럼처럼 사용하기 위해 주로 사용**

#### \*\*문법

```

SQL1 *
1 SELECT * | 컬럼명 | 표현식,
2   (SELECT * | 컬럼명 | 표현식
3    FROM 테이블명 또는 뷰명
4    ... )
5   FROM 테이블명 또는 뷰명;

```

### 2) 인라인뷰

- FROM 절에 사용하는 서브쿼리
- **서브쿼리 결과를 테이블처럼 사용하기 위해 주로 사용**

#### \*\*문법

```

SQL1 *
1 SELECT * | 컬럼명 | 표현식
2   FROM (SELECT * | 컬럼명 | 표현식
3          FROM 테이블명 또는 뷰명
4          ... );

```

### 3) WHERE 절 서브쿼리

- 가장 일반적인 서브쿼리
- 비교 상수 자리에 값을 전달하기 위한 목적으로 주로 사용(상수항의 대체)
- 반환되는 데이터의 형태에 따라 단일행 서브쿼리, 다중행 서브쿼리, 다중컬럼 서브쿼리, (상호)연관 서브쿼리로 구분

## \*\*문법

```
SQL1 *
1 SELECT * | 컬럼명 | 표현식
2   FROM 테이블명 또는 뷔명
3 WHERE 조건대상 연산자 (SELECT * | 컬럼명 | 표현식
4       FROM 테이블명 또는 뷔명
5           ...);
```

## ● WHERE 절 서브쿼리 종류

## 1) 단일행 서브쿼리

- 서브쿼리 결과가 1개의 행만 리턴되는 형태

연산자	의 미
=	같다
<>	같지 않다
>	크다
>=	크거나 같다
<	작다
<=	작거나 같다

예제) EMP 테이블에서 전체 직원의 평균 급여보다 높은 급여를 받는 직원의 정보 출력

## STEP1) 비교대상(전체 직원 급여 평균) 확인

```
SQL1 *
1 ▶ SELECT AVG(SAL)
2   FROM EMP;
```

Result

AVG(SAL)
1 2073.214285714285714285714285714286

## STEP2) 메인쿼리의 비교 상수 자리에 서브쿼리 결과 전달

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL
2   FROM EMP
3 WHERE SAL > (SELECT AVG(SAL)
4                   FROM EMP);
```

Result

EMPNO	ENAME	SAL
1 7566	JONES	2975
2 7698	BLAKE	2850
3 7782	CLARK	2450
4 7788	SCOTT	3000
5 7839	KING	5000
6 7902	FORD	3000

## 2) 다중행 서브쿼리

- 서브쿼리 결과가 여러 행을 리턴하는 형태
- '=', '>', '<'와 같은 비교 연산자 사용 불가(여러 값이랑 비교할 수 없는 연산자들)
- 서브쿼리 결과를 한 행으로 요약하거나 다중행 서브쿼리 연산자를 사용하여 해결

### \*\* 다중행 서브쿼리 연산자

연산자	의미
IN	같은 값을 찾음
>ANY	최소값을 반환함
<ANY	최대값을 반환함
<ALL	최소값을 반환함
>ALL	최대값을 반환함
EXISTS	서브쿼리가 하나 이상의 결과 반환시 TRUE
NOT EXISTS	서브쿼리가 결과를 반환하지 않으면 TRUE

예제) IN 연산자를 사용하여 서울에 위치한 부서 소속의 직원 이름 출력하기

&lt;EMP3&gt;

	NAME	SAL	DEPTNO
1	DAVID	2500	10
2	JANE	3000	20
3	ALICE	2800	30
4	BOB	2200	10
5	OLIVIA	3500	20
6	SMITH	3500	40

&lt;DEPT3&gt;

	DEPTNO	DNAME	LOC
1	10	HR	SEOUL
2	20	IT	SUWON
3	30	SALES	BUSAN
4	40	MARKETING	SEOUL

```
SQL1 *
1 > SELECT NAME
2   FROM EMP3
3   WHERE DEPTNO = (SELECT DEPTNO
4     FROM DEPT3
5     WHERE LOC = 'SEOUL');
```

Result  
ORA-01427: 단일 행 하위 질의에 2개 이상의 행이 리턴되었습니다.

### \*\* 해결 := 연산자 대신 IN 연산자로 변경

```
SQL1 *
1 > SELECT NAME
2   FROM EMP3
3   WHERE DEPTNO IN (SELECT DEPTNO
4     FROM DEPT3
5     WHERE LOC = 'SEOUL');
```

NAME
1 BOB
2 DAVID
3 SMITH

### ● ALL 과 ANY 비교

- > ALL(2000, 3000) : 최대값(3000)보다 큰 행들 반환
- < ALL(2000, 3000) : 최소값(2000)보다 작은 행들 반환
- > ANY(2000, 3000) : 최소값(2000)보다 큰 행들 반환
- < ANY(2000, 3000) : 최대값(3000)보다 작은 행들 반환

### 예제) 다중행 서브쿼리 연산자 오류

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL
2   FROM EMP
3   □ WHERE SAL > (SELECT SAL
4     FROM EMP
5       WHERE DEPTNO = 10);
```

Result

ORA-01427: 단일 행 하위 질의에 2개 이상의 행이 리턴되었습니다.

### \*\* 해결 1 : 서브쿼리 결과가 하나의 행으로 출력되도록 변경

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL
2   FROM EMP
3   □ WHERE SAL > (SELECT MIN(SAL)
4     FROM EMP
5       WHERE DEPTNO = 10);
```

Result

	EMPNO	ENAME	SAL
1	7499	ALLEN	1600
2	7566	JONES	2975
3	7698	BLAKE	2850
4	7782	CLARK	2450
5	7788	SCOTT	3000
6	7839	KING	5000
7	7844	TURNER	1500
8	7902	FORD	3000

### \*\* 해결 2 : 다중행 서브쿼리 연산자로 변경

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL
2   FROM EMP
3   □ WHERE SAL > ANY(SELECT SAL
4     FROM EMP
5       WHERE DEPTNO = 10);
```

Result

	EMPNO	ENAME	SAL
1	7839	KING	5000
2	7902	FORD	3000
3	7788	SCOTT	3000
4	7566	JONES	2975
5	7698	BLAKE	2850
6	7782	CLARK	2450
7	7499	ALLEN	1600
8	7844	TURNER	1500

## ● EXISTS / NOT EXISTS 연산자

- 서브쿼리 조건에 따라 메인쿼리의 출력 결과가 달라짐
- 서브쿼리 SELECT 절의 출력 형태는 중요하지 않음
- 연관 서브쿼리를 사용하여 연관 조건의 결과에 따라 메인쿼리 출력을 제한할 수 있음

### 1. EXISTS

- 서브쿼리 결과가 하나라도 존재하면 메인쿼리 출력

### 2. NOT EXISTS

- 서브쿼리 결과가 존재하지 않으면 메인쿼리 데이터 출력

### 예제) EXISTS 사용

```
SQL1 *
1 ▷ SELECT ENAME, SAL
2   FROM EMP
3   WHERE DEPTNO = 10
4   AND EXISTS (SELECT 1
5     FROM DEPT);
6
```

Result

	ENAME	SAL
1	CLARK	2450
2	KING	5000
3	MILLER	1300

```
SQL1 *
1 ▷ SELECT ENAME, SAL
2   FROM EMP
3   WHERE DEPTNO = 10
4   AND EXISTS (SELECT 1
5     FROM DEPT
6     WHERE 1=2);
```

Result

	ENAME	SAL
--	-------	-----

### 예제) NOT EXISTS 사용

```
SQL1 *
1 ▷ SELECT ENAME, SAL
2   FROM EMP
3   WHERE DEPTNO = 10
4   AND NOT EXISTS (SELECT 1
5     FROM DEPT);
6
```

Result

	ENAME	SAL
--	-------	-----

```
SQL1 *
1 ▷ SELECT ENAME, SAL
2   FROM EMP
3   WHERE DEPTNO = 10
4   AND NOT EXISTS (SELECT 1
5     FROM DEPT
6     WHERE 1=2);
```

Result

	ENAME	SAL
1	CLARK	2450
2	KING	5000
3	MILLER	1300

### 3) 다중컬럼 서브쿼리

- 서브쿼리 결과가 여러 컬럼을 리턴하는 형태
- 메인쿼리와 비교하는 컬럼이 2개 이상인 형태
- 대소 비교 조건 전달 불가(두 값을 동시에 묶어 대소 비교를 할 수 없으므로)

예제) EMP 테이블에서 부서별 최대 급여자 확인

### STEP1) 부서별 최대 급여 확인

```
SQL1 *
1 ▶ SELECT DEPTNO, MAX(SAL)
2   FROM EMP
3   GROUP BY DEPTNO;
```

Result

DEPTNO	MAX(SAL)
1	30
2	20
3	10

### STEP2) 위 결과를 메인쿼리의 비교 상수로 전달

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL, DEPTNO
2   FROM EMP
3   WHERE (DEPTNO, SAL) IN (SELECT DEPTNO, MAX(SAL)
4                             FROM EMP
5                             GROUP BY DEPTNO);
```

Result

EMPNO	ENAME	SAL	DEPTNO
1	BLAKE	2850	30
2	FORD	3000	20
3	SCOTT	3000	20
4	KING	5000	10

☞ 부서별 최대 급여는 여러 행을 리턴하기 때문에 IN 연산자를 사용해야 함(= 사용 시 오류 발생)

## 4) 상호연관 서브쿼리

- 메인쿼리와 서브쿼리 간의 비교를 수행하는 형태
- 비교할 집단이나 조건은 서브쿼리에 명시되어야 함
- 서브쿼리가 메인쿼리의 컬럼을 참조하고 있기 때문에, 메인쿼리 실행 시 서브쿼리가 항상 실행되는 형태

예제) 소속 부서의 평균급여보다 높은 급여를 받는 사원 정보 출력

**\*\* 에러 발생 : 다중컬럼 서브쿼리는 동시에 두 컬럼을 비교할 수 없음**

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL, DEPTNO
2   FROM EMP
3   WHERE (DEPTNO, SAL) > (SELECT DEPTNO, AVG(SAL)
4                             FROM EMP
5                             GROUP BY DEPTNO);
```

Result

ORA-01796: 연산자의 지정이 부적합합니다

\*\* 해결) 대소 비교할 컬럼을 메인쿼리에, 일치 조건을 서브쿼리에 전달

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL, DEPTNO
2   FROM EMP E1
3   WHERE SAL > (SELECT AVG(SAL)
4                   FROM EMP E2
5                   WHERE E1.DEPTNO = E2.DEPTNO
6                   GROUP BY DEPTNO);
```

Result

	EMPNO	ENAME	SAL	DEPTNO
1	7499	ALLEN	1600	30
2	7566	JONES	2975	20
3	7698	BLAKE	2850	30
4	7788	SCOTT	3000	20
5	7839	KING	5000	10
6	7902	FORD	3000	20

- ☞ 메인쿼리와 결과적으로 비교해야 할 컬럼은 SAL 이므로, 메인쿼리에는 SAL 만 명시 DEPTNO 정보가 서로 일치하는 경우만 SAL 에 대한 비교를 해야 하므로 DEPTNO 에 대한 조건은 서브쿼리로 전달
- ☞ 메인쿼리에는 E.DEPTNO = D.DEPTNO 조건 사용불가

\*\* 상호연관 서브쿼리 연산 순서

- 1) 메인쿼리 테이블 읽기(한 행씩)
  - 2) 메인쿼리 WHERE 절 확인(SAL 확인)
  - 3) 서브쿼리 테이블 읽기(FROM 절부터)
  - 4) 서브쿼리 WHERE 절 확인 -> 다시 메인쿼리의 DEPTNO(E1.DEPTNO) 요구
  - 5) 서브쿼리에서 E1.DEPTNO 값과 같은 대상을 찾아 AVG(SAL) 연산
  - 6) SAL > AVG(SAL) 조건을 만족하는 행만 추출
- \* 상호연관 서브쿼리 사용 시 GROUP BY 생략 가능

예제) EXISTS / NOT EXISTS 연관 서브쿼리

<EXISTS_TAB1>		
	NO	NAME
1	1	AMERICANO
2	2	LATTE
3	3	MOCHA
4	4	CAPPUCCINO

<EXISTS_TAB2>		
	NO	SALE
1	2	10
2	4	15

SQL1 \*

```

1 SELECT *
2   FROM EXISTS_TAB1 E1
3 WHERE EXISTS (SELECT 'X'
4                   FROM EXISTS_TAB2 E2
5                 WHERE E1.NO = E2.NO);

```

Result

NO	NAME	PRICE
1	LATTE	2000
2	CAPPUCCINO	3500

- ☞ 연관 서브쿼리는 메인쿼리의 각 행이 실행될 때마다 서브쿼리가 실행되며, E1.NO = E2.NO 조건의 참/거짓 여부를 판별하여 메인쿼리 결과가 선택됨

### ● 인라인뷰(Inline View)

- 쿼리 안의 뷰의 형태로, 테이블처럼 조회할 데이터를 정의하기 위해 사용
- 테이블명이 존재하지 않기 때문에 다른 테이블과 조인 시 반드시 테이블 별칭을 명시해야 함  
(조인 없이 단독으로 사용하는 경우 불필요)
- 서브쿼리에서 출력되는 결과를 메인 쿼리의 어느 절에서도 사용할 수 있음
- 인라인 뷰의 결과는 메인쿼리 테이블과 조인할 목적으로 주로 사용
- 모든 비교 연산자 사용 가능

예제) EMP 테이블에서 부서별 최대 급여자를 출력하되, 최대 급여와 함께 출력

SQL1 \*

```

1 SELECT E.EMPNO, E.ENAME, E.SAL, I.MAX_SAL
2   FROM EMP E, (SELECT DEPTNO, MAX(SAL) AS MAX_SAL
3                  FROM EMP
4                 GROUP BY DEPTNO) I
5 WHERE E.DEPTNO = I.DEPTNO
6   AND E.SAL = I.MAX_SAL;

```

Result

EMPNO	ENAME	SAL	MAX_SAL
7698	BLAKE	2850	2850
7788	SCOTT	3000	3000
7839	KING	5000	5000
7902	FORD	3000	3000

예제) EMP 테이블에서 부서별로 해당 부서의 평균 급여보다 높은 급여자를 출력하되, 평균 급여와 함께 출력

```
SQL1 *
1 ▷ SELECT E.EMPNO, E.ENAME, E.SAL, I.AVG_SAL
2   FROM EMP E, (SELECT DEPTNO, AVG(SAL) AS AVG_SAL
3                  FROM EMP
4                 GROUP BY DEPTNO) I
5   WHERE E.DEPTNO = I.DEPTNO
6     AND E.SAL > I.AVG_SAL;
```

Result

	EMPNO	ENAME	SAL	AVG_SAL
1	7499	ALLEN	1600	1566.666...
2	7566	JONES	2975	2175
3	7698	BLAKE	2850	1566.666...
4	7788	SCOTT	3000	2175
5	7839	KING	5000	2916.666...
6	7902	FORD	3000	2175

### ● 스칼라 서브쿼리

- SELECT 절에 사용하는 쿼리로, 마치 하나의 컬럼처럼 표현하기 위해 사용(단, 하나의 출력 대상만 표현 가능)
- 각 행마다 스칼라 서브쿼리 결과가 하나여야 함(단일행 서브쿼리 형태)
- 조인의 대체 연산
- 스칼라 서브쿼리를 사용한 조인 시 OUTER JOIN 이 기본(조인 조건에 일치하는 대상이 없어도 생략되지 않고 NULL로 출력됨)

예제) EMP 의 각 직원의 사번, 이름과 부서 이름을 출력(부서 이름을 스칼라 서브쿼리로 표현)

```
SQL1 *
1 ▷ SELECT EMPNO, ENAME,
2   (SELECT DNAME
3    FROM DEPT D
4    WHERE D.DEPTNO = E.DEPTNO) AS DNAME
5   FROM EMP E
6  WHERE DEPTNO = 10;
```

Result

	EMPNO	ENAME	DNAME
1	7782	CLARK	ACCOUNTING
2	7839	KING	ACCOUNTING
3	7934	MILLER	ACCOUNTING

예제) EMP 테이블에서 각 직원의 사번, 이름, 부서번호, 급여와 함께 급여 총합을 출력  
(총합을 스칼라 서브쿼리로 표현)

```
SQL1 *
1 SELECT EMPNO, ENAME, DEPTNO, SAL,
2      (SELECT SUM(SAL)
3       FROM EMP) AS SUM_SAL
4   FROM EMP;
```

Result

	EMPNO	ENAME	DEPTNO	SAL	SUM_SAL
1	7369	SMITH	20	800	29025
2	7499	ALLEN	30	1600	29025
3	7521	WARD	30	1250	29025
4	7566	JONES	20	2975	29025
5	7654	MARTIN	30	1250	29025
6	7698	BLAKE	30	2850	29025
7	7782	CLARK	10	2450	29025
8	7788	SCOTT	20	3000	29025
9	7839	KING	10	5000	29025
10	7844	TURNER	30	1500	29025
11	7876	ADAMS	20	1100	29025
12	7900	JAMES	30	950	29025
13	7902	FORD	20	3000	29025
14	7934	MILLER	10	1300	29025

예제) 스칼라 서브쿼리의 아우터 조인

```
SQL1 *
1 SELECT E1.ENAME AS 사원명,
2      (SELECT E2.ENAME
3       FROM EMP E2
4      WHERE E1.MGR = E2.EMPNO) AS 상위관리자명
5   FROM EMP E1;
```

Result

	사원명	상위관리자명
1	SMITH	FORD
2	ALLEN	BLAKE
3	WARD	BLAKE
4	JONES	KING
5	MARTIN	BLAKE
6	BLAKE	KING
7	CLARK	KING
8	SCOTT	JONES
9	KING	

이하 생략

☞ KING의 경우 MGR 컬럼 값이 NULL 이므로 MGR = EMPNO에 만족하는 E2.ENAME 값이 없지만,  
스칼라 서브쿼리는 무조건 메인쿼리 절이 출력하는 대상에 대해 항상 값을 리턴해야 하므로 생략되지 않고  
NULL로 출력됨

### ● 서브쿼리 주의 사항

- 특별한 경우(TOP-N 분석 등)를 제외하고는 서브쿼리 절에 ORDER BY 절 사용 불가
- 단일행 서브쿼리와 다중행 서브쿼리에 따라 연산자의 선택이 중요

예제) 서브쿼리에 ORDER BY 전달 시 에러 발생

```
SQL1 *
1 SELECT *
2   FROM EMP
3 WHERE SAL IN (SELECT SAL
4                  FROM EMP
5                  WHERE DEPTNO = 10
6                  ORDER BY SAL);
```

Result

Error: # 907, ORA-00907: 누락된 우괄호

## 2-10. 집합 연산자

### ● 집합 연산자

- SELECT 문 결과를 하나의 집합으로 간주, 그 집합에 대한 합집합, 교집합, 차집합 연산
- SELECT 문과 SELECT 문 사이에 집합 연산자 정의
- 두 집합의 컬럼이 동일하게 구성되어야 함(각 컬럼의 데이터 타입과 순서 일치 필요)
- 전체 집합의 데이터타입과 컬럼명은 첫번째 집합에 의해 결정됨

### ● 합집합

- 두 집합의 총합(전체) 출력
- UNION 과 UNION ALL 사용 가능

#### 1) UNION

- 중복된 데이터는 한 번만 출력
- 중복된 데이터를 제거하기 위해 내부적으로 정렬 수행
- 중복된 데이터가 없을경우는 UNION 사용 대신 UNION ALL 사용 권고(불필요한 정렬 발생할 수 있으므로)

#### 2) UNION ALL

- 중복된 데이터도 전체 출력

예제) 10 번 부서 소속이 아닌 직원 정보와 20 번 소속 직원 정보가 각각 분리되어있다 가정할 때 두 집합의 합집합

### UNION 결과)

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, DEPTNO
2   FROM EMP
3 WHERE DEPTNO != 10
4 UNION
5 SELECT EMPNO, ENAME, DEPTNO
6   FROM EMP
7 WHERE DEPTNO = 20;
```

Result

	EMPNO	ENAME	DEPTNO
1	7369	SMITH	20
2	7499	ALLEN	30
3	7521	WARD	30
4	7566	JONES	20
5	7654	MARTIN	30
6	7698	BLAKE	30
7	7788	SCOTT	20
8	7844	TURNER	30
9	7876	ADAMS	20
10	7900	JAMES	30
11	7902	FORD	20

☞ 중복 데이터가 없음

### UNION ALL 결과)

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, DEPTNO
2   FROM EMP
3 WHERE DEPTNO != 10
4 UNION ALL
5 SELECT EMPNO, ENAME, DEPTNO
6   FROM EMP
7 WHERE DEPTNO = 20;
```

Result

	EMPNO	ENAME	DEPTNO
1	7369	SMITH	20
2	7499	ALLEN	30
3	7521	WARD	30
4	7566	JONES	20
5	7654	MARTIN	30
6	7698	BLAKE	30
7	7788	SCOTT	20
8	7844	TURNER	30
9	7876	ADAMS	20
10	7900	JAMES	30
11	7902	FORD	20
12	7369	SMITH	20
13	7566	JONES	20
14	7788	SCOTT	20
15	7876	ADAMS	20
16	7902	FORD	20

☞ 중복 데이터도 모두 출력

## ● 교집합

- 두 집합 사이에 INTERSECT
- 두 집합의 교집합(공통으로 있는 행) 출력

예제) 10 번 부서 정보와 20 번 부서 정보가 각각 분리되어있다고 가정할 때

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL, DEPTNO
2   FROM EMP
3 WHERE DEPTNO != 10
4 INTERSECT
5 SELECT EMPNO, ENAME, SAL, DEPTNO
6   FROM EMP
7 WHERE DEPTNO != 20;
```

Result

	EMPNO	ENAME	SAL	DEPTNO
1	7499	ALLEN	1600	30
2	7521	WARD	1250	30
3	7654	MARTIN	1250	30
4	7698	BLAKE	2850	30
5	7844	TURNER	1500	30
6	7900	JAMES	950	30

☞ 부서번호가 10 번이 아닌 집합은 20, 30 번 이고, 20 번이 아닌 부서원은 10, 30 번 부서원이므로 두 집합의 교집합인 30 번 부서원 정보만 출력됨

## ● 차집합

- 두 집합 사이에 MINUS 전달
- 두 집합의 차집합(한 쪽 집합에만 존재하는 행) 출력
- A-B 와 B-A는 다르므로 집합의 순서 주의!

예제) 10 번 부서 정보와 20 번 부서 정보가 각각 분리되어있다고 가정할 때

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL, DEPTNO
2   FROM EMP
3 WHERE DEPTNO != 10
4 MINUS
5 SELECT EMPNO, ENAME, SAL, DEPTNO
6   FROM EMP
7 WHERE DEPTNO = 20;
```

Result

	EMPNO	ENAME	SAL	DEPTNO
1	7499	ALLEN	1600	30
2	7521	WARD	1250	30
3	7654	MARTIN	1250	30
4	7698	BLAKE	2850	30
5	7844	TURNER	1500	30
6	7900	JAMES	950	30

☞ 부서번호가 10 번이 아닌 (20,30)번인 집합에서 20 번 집합을 빼면 30 번 부서원 집합만 출력됨

● 집합 연산자 사용시 주의 사항

1. 두 집합의 컬럼 수 일치
2. 두 집합의 컬럼 순서 일치
3. 두 집합의 각 컬럼의 데이터 타입 일치
4. 각 컬럼의 사이즈는 달라도 됨
5. 개별 SELECT 문에 ORDER BY 전달 불가(GROUP BY 전달 가능)

예제) 두 집합의 컬럼의 데이터타입이 다른 경우 에러 발생

아래와 같은 EMP\_T1 테이블이 있다고 가정, EMP 와의 합집합 출력

SQL1 \*

```
1 ▶ DESC EMP_T1;
```

Result

Column	Nullable	Type
1 EMPNO	NOT NULL	NUMBER(4)
2 ENAME		VARCHAR2(10)
3 JOB		VARCHAR2(9)
4 MGR		NUMBER(4)
5 HIREDATE		DATE
6 SAL		NUMBER(7,2)
7 COMM		NUMBER(7,2)
8 DEPTNO		NUMBER(2)

☞ EMP 테이블의 각 컬럼별 데이터 타입

SQL1 \*

```
1 ▶ DESC EMP_T1;
```

Result

Column	Nullable	Type
1 EMPNO		VARCHAR2(40)
2 ENAME		VARCHAR2(10)
3 SAL		NUMBER(7,2)
4 DEPTNO		NUMBER(2)

☞ EMP\_T1 테이블의 각 컬럼별 데이터 타입

**\*\* 예러)**

SQL1 \*

```
1 ▶ SELECT EMPNO, ENAME, SAL, DEPTNO
2   FROM EMP
3 WHERE DEPTNO = 10
4 UNION
5 SELECT EMPNO, ENAME, SAL, DEPTNO
6   FROM EMP_T1;
```

Result

ORA-01790: 대응하는 식과 같은 데이터 유형이어야 합니다

☞ 에러 발생(EMPNO 컬럼 데이터 타입 서로 다름)

## \*\* 해결)

```
SQL1 *
1 ▷ SELECT EMPNO, ENAME, SAL, DEPTNO
2   FROM EMP
3  WHERE DEPTNO = 10
4  UNION
5  SELECT TO_NUMBER(EMPNO),
6         ENAME, SAL, DEPTNO
7   FROM EMP_T1;
```

Result

	EMPNO	ENAME	SAL	DEPTNO
1	7369	SMITH	800	20
2	7499	ALLEN	1600	30
3	7521	WARD	1250	30
4	7566	JONES	2975	20
5	7654	MARTIN	1250	30
6	7698	BLAKE	2850	30
7	7782	CLARK	2450	10
8	7788	SCOTT	3000	20
9	7839	KING	5000	10
10	7844	TURNER	1500	30
11	7876	ADAMS	1100	20
12	7900	JAMES	950	30
13	7902	FORD	3000	20
14	7934	MILLER	1300	10

☞ 두 집합의 EMPNO 컬럼의 데이터 타입이 다르므로 한 쪽으로 맞춰줄 필요 있음

## 예제) 집합연산자와 ORDER BY의 사용

```
SQL1 *
1 ▷ SELECT ENAME, SAL
2   FROM EMP
3  WHERE DEPTNO = 10
4  ORDER BY SAL
5  UNION ALL
6  SELECT ENAME, SAL
7   FROM EMP
8  WHERE DEPTNO = 20
9  ORDER BY SAL;
```

Result  
ORA-00933: SQL 명령어가 올바르게 종료되지 않았습니다

☞ 개별 쿼리에는 ORDER BY를 전달할 수 없음

```
SQL1 *
1 ▷ (SELECT ENAME, SAL
2   FROM EMP E1
3  WHERE DEPTNO = 10
4  UNION ALL
5  SELECT ENAME, SAL
6   FROM EMP E2
7  WHERE DEPTNO = 20)
8  ORDER BY SAL;
```

Result

	ENAME	SAL
1	SMITH	800
2	ADAMS	1100
3	MILLER	1300
4	CLARK	2450
5	JONES	2975
6	FORD	3000
7	SCOTT	3000
8	KING	5000

☞ 집합 연산자 수행 후 전체 결과에 ORDER BY 절 전달 가능(괄호 생략 가능)

## 2-11. 그룹 함수

### ● 그룹함수

- 숫자함수 중 여러 값을 전달하여 하나의 요약 값을 출력하는 다중행 함수
- 수학/통계 함수들(기술통계 함수)
- GROUP BY 절에 의해 그룹별 연산 결과를 리턴 함
- [반드시 한 컬럼만 전달](#)
- [NULL은 무시하고 연산](#)

### ● COUNT

- 테이블의 행의 수를 세는 함수
- 인수 : \* 또는 하나의 표현식
- 문자, 숫자, 날짜 컬럼 모두 전달 가능
- NULL은 세지 않는다
- COUNT(\*)은 항상 모든 행의 수를 출력

#### \*\*문법

SQL1 *	1 COUNT(대상)
--------	-------------

#### 예제) 각 컬럼의 COUNT 결과

SQL1 *	1 SELECT COUNT(*), 2 COUNT(EMPNO), 3 COUNT(COMM) 4 FROM EMP;									
Result										
<a href="#">Grid Result</a> <a href="#">Server Output</a> <a href="#">Text Output</a> <a href="#">Explain Plan</a> <a href="#">Statistics</a>										
<a href="#">Grid Result</a> <a href="#">Server Output</a> <a href="#">Text Output</a> <a href="#">Explain Plan</a> <a href="#">Statistics</a>										
<table border="1"> <thead> <tr> <th>COUNT(*)</th> <th>COUNT(EMPNO)</th> <th>COUNT(COMM)</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>14</td> <td>14</td> </tr> <tr> <td>1</td> <td>14</td> <td>4</td> </tr> </tbody> </table>		COUNT(*)	COUNT(EMPNO)	COUNT(COMM)	1	14	14	1	14	4
COUNT(*)	COUNT(EMPNO)	COUNT(COMM)								
1	14	14								
1	14	4								

☞ COMM 은 NULL 을 포함한 컬럼이므로 전체 행의 수와 다른 값이 출력됨

#### 예제) NULL 로만 구성된 테이블의 COUNT 결과

<NULL\_TAB1>

	NO	NAME
1		
2		

```
SQL1 *
1 SELECT COUNT(*)
2 FROM NULL_TAB1;
```

Result

COUNT(*)
2

☞ COUNT(\*) 결과는 항상 전체 건수를 리턴한다.

(단, 모든 컬럼 값이 NULL인 행은 모델링 과정의 인스턴스 규칙에 어긋나기 때문에 현실적으로 존재하기 힘든 형태)

## ● SUM

- 총합 출력
- 숫자 컬럼만 전달 가능

예제) 급여의 전체 총합

```
SQL1 *
1 SELECT SUM(SAL)
2   FROM EMP;
```

Result

SUM(SAL)
29025

## ● AVG

- 평균 출력
- 숫자 컬럼만 전달 가능
- NULL을 제외한 대상의 평균을 리턴하므로 전체 대상 평균 연산 시 주의

\*\* 문법

```
SQL1 *
1 AVG(대상)
```

예제) 평균 계산 결과

```
SQL1 *
1 SELECT AVG(COMM),
2        SUM(COMM) / COUNT(EMPNO) AS AVG2,
3        AVG(NVL(COMM, 0)) AS AVG3
4   FROM EMP;
```

Result

AVG(COMM)	AVG2	AVG3
550	157.142...	157.142...

☞ AVG를 사용하면 널을 제외한 나머지에 대한 평균(4명에 대한) 리턴, 공식에 의해 직접 계산한 평균은 14명에 대한 평균

☞ NVL 함수를 사용하여 널을 0으로 치환 후 평균을 구하면 총 14명에 대한 평균과 같아짐

## ● MIN / MAX

- 최소, 최대 출력
- 날짜, 숫자, 문자 모두 가능(오름차순 순서대로 최소, 최대 출력)

\*\* 문법

SQL1 *	
1	MIN(대상) / MAX(대상)

### 예제) 각 컬럼의 최대, 최소

SQL1 *													
1	SELECT MIN(ENAME), MAX(ENAME), MIN(SAL), MAX(SAL), MIN(HIREDATE), MAX(HIREDATE) FROM EMP;												
Result													
<a href="#">Grid Result</a> <a href="#">Server Output</a> <a href="#">Text Output</a> <a href="#">Explain Plan</a> <a href="#">Statistics</a>													
<table border="1"> <thead> <tr> <th>MIN(ENAME)</th><th>MAX(ENAME)</th><th>MIN(SAL)</th><th>MAX(SAL)</th><th>MIN(HIREDATE)</th><th>MAX(HIREDATE)</th></tr> </thead> <tbody> <tr> <td>ADAMS</td><td>WARD</td><td>800</td><td>5000</td><td>1980/12/17 00:00:00</td><td>1987/05/23 00:00:00</td></tr> </tbody> </table>		MIN(ENAME)	MAX(ENAME)	MIN(SAL)	MAX(SAL)	MIN(HIREDATE)	MAX(HIREDATE)	ADAMS	WARD	800	5000	1980/12/17 00:00:00	1987/05/23 00:00:00
MIN(ENAME)	MAX(ENAME)	MIN(SAL)	MAX(SAL)	MIN(HIREDATE)	MAX(HIREDATE)								
ADAMS	WARD	800	5000	1980/12/17 00:00:00	1987/05/23 00:00:00								

## ● VARIANCE / STDDEV

- 분산과 표준편차
- 표준편차는 분산의 루트값

\*\* 문법

SQL1 *	
1	VARIANCE(대상) / STDDEV(대상)

### 예제) 분산과 표준편차

SQL1 *					
1	SELECT VARIANCE(SAL), STDDEV(SAL) FROM EMP;				
Result					
<a href="#">Grid Result</a> <a href="#">Server Output</a> <a href="#">Text Output</a> <a href="#">Explain Plan</a> <a href="#">Statistics</a>					
<table border="1"> <thead> <tr> <th>VARIANCE(SAL)</th><th>STDDEV(SAL)</th></tr> </thead> <tbody> <tr> <td>1398313.873...</td><td>1182.503...</td></tr> </tbody> </table>		VARIANCE(SAL)	STDDEV(SAL)	1398313.873...	1182.503...
VARIANCE(SAL)	STDDEV(SAL)				
1398313.873...	1182.503...				

## ● GROUP BY FUNCTION

- GROUP BY 절에 사용하는 함수
- 여러 GROUP BY 결과를 동시에 출력(합집합)하는 기능
- 그룹핑 할 그룹을 정의(전체 소계 등)

예제) 본 GROUP BY 기능 : 그룹별 연산값만 출력되므로 전체 소계와 함께 출력될 수 없음

```
SQL1 *
1 SELECT DEPTNO, SUM(SAL)
2   FROM EMP
3  GROUP BY DEPTNO;
```

Result

DEPTNO	SUM(SAL)
1	30 9400
2	20 10875
3	10 8750

### 1. GROUPING SETS(A, B, ...)

- A 별, B 별 그룹 연산 결과 출력
- 나열 순서 중요하지 X
- 기본 출력에 전체 총계는 출력되지 X
- NULL 혹은 () 사용하여 전체 총합 출력 가능

예제) DEPTNO 별 SAL 의 총합 결과와 JOB 별 SAL 의 총합 결과의 합집합

```
SQL1 *
1 SELECT DEPTNO, JOB, SUM(SAL)
2   FROM EMP
3  GROUP BY GROUPING SETS(DEPTNO, JOB);
```

Result

DEPTNO	JOB	SUM(SAL)
1	CLERK	4150
2	SALESMAN	5600
3	PRESIDENT	5000
4	MANAGER	8275
5	ANALYST	6000
6	30	9400
7	20	10875
8	10	8750

☞ GROUPING SETS 에 나열한 대상에 대해 각 GROUP BY 의 결과를 출력해 줌

### \*\* UNION ALL로 대체 가능

SQL1 \*

```

1 SELECT DEPTNO, NULL AS JOB, SUM(SAL)
2   FROM EMP
3  GROUP BY DEPTNO
4 UNION ALL
5 SELECT NULL, JOB, SUM(SAL)
6   FROM EMP
7  GROUP BY JOB;

```

Result

DEPTNO	JOB	SUM(SAL)
1	30	9400
2	20	10875
3	10	8750
4	CLERK	4150
5	SALESMAN	5600
6	PRESIDENT	5000
7	MANAGER	8275
8	ANALYST	6000

### 예제) 부서별 급여 총합과, 업무별 급여 총합, 그리고 전체 급여의 합을 출력

SQL1 \*

```

1 SELECT DEPTNO, JOB, SUM(SAL)
2   FROM EMP
3  GROUP BY GROUPING SETS(DEPTNO, JOB, ());

```

Result

DEPTNO	JOB	SUM(SAL)
1	CLERK	4150
2	SALESMAN	5600
3	PRESIDENT	5000
4	MANAGER	8275
5	ANALYST	6000
6	10	8750
7	20	10875
8	30	9400
9		29025

## 2. ROLLUP(A,B)

- A 별, (A,B) 별, 전체 그룹 연산 결과 출력
- 나열 대상의 순서가 중요함
- 기본적으로 전체 총 계가 출력됨

예제) ROLLUP(DEPTNO, JOB) -> DEPTNO 별, (DEPTNO, JOB)별, 전체 연산 결과 출력

```
SQL1 *
1 ▷ SELECT DEPTNO, JOB, SUM(SAL)
2   FROM EMP
3   GROUP BY ROLLUP(DEPTNO, JOB);
```

Result

DEPTNO	JOB	SUM(SAL)
1	10 CLERK	1300
2	10 MANAGER	2450
3	10 PRESIDENT	5000
4	10	8750
5	20 CLERK	1900
6	20 ANALYST	6000
7	20 MANAGER	2975
8	20	10875
9	30 CLERK	950
10	30 MANAGER	2850
11	30 SALESMAN	5600
12	30	9400
13		29025

DEPTNO, JOB별 그룹 연산  
DEPTNO별 그룹 선언  
전체 총 합 결과

\*\* UNION ALL로 대체 가능

```
SQL1 *
1 ▷ SELECT DEPTNO, JOB, SUM(SAL)
2   FROM EMP
3   GROUP BY DEPTNO, JOB
4   UNION ALL
5   SELECT DEPTNO, NULL, SUM(SAL)
6   FROM EMP
7   GROUP BY DEPTNO
8   UNION ALL
9   SELECT NULL, NULL, SUM(SAL)
10  FROM EMP;
```

Result

DEPTNO	JOB	SUM(SAL)
1	20 CLERK	1900
2	30 SALESMAN	5600
3	20 MANAGER	2975
4	30 CLERK	950
5	10 PRESIDENT	5000
6	30 MANAGER	2850
7	10 CLERK	1300
8	10 MANAGER	2450
9	20 ANALYST	6000
10	30	9400
11	20	10875
12	10	8750
13		29025

### 3. CUBE(A,B)

- A 별, B 별, (A,B)별, 전체 그룹 연산 결과 출력됨

- 그룹으로 묶을 대상의 나열 순서 중요하지 않음
- 기본적으로 전체 총 계가 출력됨

## 예제) DEPTNO 별, JOB 별, (DEPTNO, JOB)별, 전체 급여의 총합 출력

SQL1 \*

```

1 ▶ SELECT DEPTNO, JOB, SUM(SAL)
2   FROM EMP
3 GROUP BY CUBE(DEPTNO, JOB);

```

Result

	DEPTNO	JOB	SUM(SAL)
1			29025
2		CLERK	4150
3		ANALYST	6000
4		MANAGER	8275
5		SALESMAN	5600
6		PRESIDENT	5000
7	10		8750
8	10	CLERK	1300
9	10	MANAGER	2450
10	10	PRESIDENT	5000
11	20		10875
12	20	CLERK	1900
13	20	ANALYST	6000
14	20	MANAGER	2975
15	30		9400
16	30	CLERK	950
17	30	MANAGER	2850
18	30	SALESMAN	5600

## UNION ALL로 대체)

SQL1 \*

```

1 ▶ SELECT NULL AS DEPTNO, NULL AS JOB, SUM(SAL)
2   FROM EMP
3 UNION ALL
4 SELECT NULL, JOB, SUM(SAL)
5   FROM EMP
6 GROUP BY JOB
7 UNION ALL
8 SELECT DEPTNO, NULL, SUM(SAL)
9   FROM EMP
10 GROUP BY DEPTNO
11 UNION ALL
12 SELECT DEPTNO, JOB, SUM(SAL)
13   FROM EMP
14 GROUP BY DEPTNO, JOB;

```

Result

	DEPTNO	JOB	SUM(SAL)
1			29025
2		CLERK	4150
3		SALESMAN	5600
4		PRESIDENT	5000
5		MANAGER	8275
6		ANALYST	6000
7	30		9400

이하 생략

### GROUPING SETS 로 대체)

SQL1 \*

```

1 SELECT DEPTNO, JOB, SUM(SAL)
2   FROM EMP
3 GROUP BY GROUPING SETS(DEPTNO, JOB, (DEPTNO, JOB), ());

```

Result

DEPTNO	JOB	SUM(SAL)
1	10 CLERK	1300
2	20 CLERK	1900
3	30 CLERK	950
4	20 ANALYST	6000
5	10 MANAGER	2450
6	20 MANAGER	2975
7	30 MANAGER	2850
8	30 SALESMAN	5600
9	10 PRESIDENT	5000
10	CLERK	4150
11	ANALYST	6000
12	MANAGER	8275
13	SALESMAN	5600
14	PRESIDENT	5000

이하 생략

## 2-12. 원도우 함수

### ● WINDOW FUNCTION

- 서로 다른 행의 비교나 연산을 위해 만든 함수
- GROUP BY 를 쓰지 않고 그룹 연산 가능
- LAG, LEAD, SUM, AVG, MIN, MAX, COUNT, RANK

#### \*\* 문법

```
SQL1 *
1 SELECT 윈도우함수([대상]) OVER([PARTITION BY 컬럼]
2   [ORDER BY 컬럼 ASC|DESC]
3   [ROWS|RANGE BETWEEN A AND B])
```

\*\* PARTITION BY 절 : 출력할 총 데이터 수 변화 없이 그룹연산 수행할 GROUP BY 컬럼

#### \*\* ORDER BY 절

- RANK의 경우 필수(정렬 컬럼 및 정렬 순서에 따라 순위 변화)
- SUM, AVG, MIN, MAX, COUNT 등은 누적값 출력 시 사용

#### \*\* ROWS|RANGE BETWEEN A AND B

- 연산 범위 설정
- ORDER BY 절 필수

\* PARTITION BY, ORDER BY, ROWS...절 전달 순서 중요(ORDER BY 를 PARTITON BY 전에 사용 불가)

### 예제) 그룹함수 오류(원도우 함수가 필요한 이유)

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL, DEPTNO, SUM(SAL)
2   FROM EMP;
```

Result  
ORA-00937: 단일 그룹의 그룹 함수가 아닙니다

☞ 전체를 출력하는 컬럼과 그룹함수 결과는 함께 출력할 수 없음

### ● 원도우 함수의 형태

- SUM, COUNT, AVG, MIN, MAX 등
- OVER 절을 사용하여 원도우 함수로 사용 가능
- 반드시 연산할 대상을 그룹함수의 입력값으로 전달

#### \*\* 문법

```
SQL1 *
1 ▶ SELECT SUM([대상]) OVER([PARTITION BY 컬럼]
2   [ORDER BY 컬럼 ASC|DESC]
3   [ROWS|RANGE BETWEEN A AND B])
```

### 1) SUM OVER()

- 전체 총합, 그룹별 총합 출력 가능

#### 예제) 각 직원 정보와 함께 급여 총합 출력

**\*\* 에러 : 각 직원 정보와 급여 총합(그룹함수 결과)을 동시에 출력 시도 시 에러 발생**

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL, DEPTNO, SUM(SAL)
2   FROM EMP;
```

Result  
ORA-00937: 단일 그룹의 그룹 함수가 아닙니다

#### \*\* 해결 1 : 서브쿼리 사용(스칼라 서브쿼리)

```
SQL1 *
1 ▶ SELECT EMPNO, ENAME, SAL, DEPTNO,
2       (SELECT SUM(SAL) FROM EMP) AS TOTAL
3     FROM EMP;
```

Result  
Grid Result Server Output Text Output Explain Plan Statistics  

	EMPNO	ENAME	SAL	DEPTNO	TOTAL
1	7369	SMITH	800	20	29025
2	7499	ALLEN	1600	30	29025
3	7521	WARD	1250	30	29025
4	7566	JONES	2975	20	29025
5	7654	MARTIN	1250	30	29025
6	7698	BLAKE	2850	30	29025
7	7782	CLARK	2450	10	29025
8	7788	SCOTT	3000	20	29025
9	7839	KING	5000	10	29025
10	7844	TURNER	1500	30	29025
11	7876	ADAMS	1100	20	29025
12	7900	JAMES	950	30	29025
13	7902	FORD	3000	20	29025
14	7934	MILLER	1300	10	29025

## \*\* 해결 2 : 윈도우 함수 사용

SQL1 \*

```

1 SELECT EMPNO, ENAME, SAL, DEPTNO,
2      SUM(SAL) OVER() AS TOTAL
3   FROM EMP;

```

Result

	EMPNO	ENAME	SAL	DEPTNO	TOTAL
1	7369	SMITH	800	20	29025
2	7499	ALLEN	1600	30	29025
3	7521	WARD	1250	30	29025
4	7566	JONES	2975	20	29025
5	7654	MARTIN	1250	30	29025
6	7698	BLAKE	2850	30	29025
7	7782	CLARK	2450	10	29025
8	7788	SCOTT	3000	20	29025
9	7839	KING	5000	10	29025
10	7844	TURNER	1500	30	29025
11	7876	ADAMS	1100	20	29025
12	7900	JAMES	950	30	29025
13	7902	FORD	3000	20	29025
14	7934	MILLER	1300	10	29025

## 2. AVG OVER() : SUM 과 동일하게 사용

예제) 각 직원 정보와 해당 직원이 속한 부서의 평균 급여 출력

SQL1 \*

```

1 SELECT EMPNO, ENAME, SAL, DEPTNO,
2      AVG(SAL) OVER(PARTITION BY DEPTNO) AS AVG_SAL
3   FROM EMP;

```

Result

	EMPNO	ENAME	SAL	DEPTNO	AVG_SAL
1	7782	CLARK	2450	10	2916.666...
2	7839	KING	5000	10	2916.666...
3	7934	MILLER	1300	10	2916.666...
4	7566	JONES	2975	20	2175
5	7902	FORD	3000	20	2175
6	7876	ADAMS	1100	20	2175
7	7369	SMITH	800	20	2175
8	7788	SCOTT	3000	20	2175
9	7521	WARD	1250	30	1566.666...
10	7844	TURNER	1500	30	1566.666...
11	7499	ALLEN	1600	30	1566.666...
12	7900	JAMES	950	30	1566.666...
13	7698	BLAKE	2850	30	1566.666...
14	7654	MARTIN	1250	30	1566.666...

## 3. MIN/MAX OVER() : SUM 과 동일하게 사용

예제) 각 직원 정보와 해당 직원이 속한 부서의 최대급여를 함께 출력

SQL1 \*

```

1 SELECT EMPNO, ENAME, SAL, DEPTNO,
2      MAX(SAL) OVER(PARTITION BY DEPTNO) AS 부서별급여총합
3      FROM EMP;

```

Result

	EMPNO	ENAME	SAL	DEPTNO	부서별급여총합
1	7782	CLARK	2450	10	5000
2	7839	KING	5000	10	5000
3	7934	MILLER	1300	10	5000
4	7566	JONES	2975	20	3000
5	7902	FORD	3000	20	3000
6	7876	ADAMS	1100	20	3000
7	7369	SMITH	800	20	3000
8	7788	SCOTT	3000	20	3000
9	7521	WARD	1250	30	2850
10	7844	TURNER	1500	30	2850
11	7499	ALLEN	1600	30	2850
12	7900	JAMES	950	30	2850
13	7698	BLAKE	2850	30	2850
14	7654	MARTIN	1250	30	2850

#### 4. COUNT : SUM 과 동일하게 사용

#### ● 윈도우 함수 연산 대상 및 범위

ROWS|RANGE BETWEEN A AND B

##### 1. 연산 대상

- 1) ROWS : 연산을 할 행을 지정
- 2) RANGE(DEFAULUT) : 연산을 할 범위를 지정

##### 2. 범위

###### 1) A : 시작점

- CURRENT ROW : 현재 행부터
- UNBOUNDED PRECEDING : 처음부터(DEFAULT)
- N PRECEDING : N 이전부터

###### 2) B : 마지막 지점 정의

- CURRENT ROW : 현재 범위까지(DEFAULT)
- UNBOUNDED FOLLOWING : 마지막까지
- N FOLLOWING : N 이후까지

##### 3. 특징

- 1) BETWEEN A : AND B 가 생략된 것으로 B 는 DEFAULT 값인 CURRENT ROW 가 적용된다.

### 예제) RANGE 기본 연산 범위

<WINDOW\_TAB1>

	NO	SAL
1	1	10
2	2	11
3	3	12
4	4	12
5	5	25
6	6	30
7	7	35

```
SQL1 *
1 SELECT NO, SAL,
2      SUM(SAL) OVER(ORDER BY SAL) AS RESULT1,
3      SUM(SAL) OVER(ORDER BY SAL
4          RANGE BETWEEN UNBOUNDED PRECEDING
5          AND CURRENT ROW) AS RESULT2
6
7 FROM WINDOW_TAB1;
```

Result

NO	SAL	RESULT1	RESULT2
1	10	10	10
2	11	21	21
3	12	45	45
4	12	45	45
5	25	70	70
6	30	100	100
7	35	135	135

☞ RANGE의 경우 값이 같으면(SAL 기준) 하나의 연산 범위로 묶어서 한 번에 연산을 수행

### 예제) RANGE 의 다양한 연산 범위

```
SQL1 *
1 SELECT NO, SAL,
2      SUM(SAL) OVER(ORDER BY SAL
3          RANGE BETWEEN 10 PRECEDING
4          AND 10 FOLLOWING) AS RESULT
5
6 FROM WINDOW_TAB1;
```

Result

NO	SAL	RESULT
1	10	45
2	11	45
3	12	45
4	12	45
5	25	90
6	30	90
7	35	90

☞ 각 행마다 각 행의 값으로부터 -10 ~ +10 범위에 있는 행들의 값을 합하여 리턴

### 예제) ROWS 연산 범위

SQL1 \*

```

1 SELECT NO, SAL,
2      SUM(SAL) OVER(ORDER BY SAL
3                   ROWS BETWEEN UNBOUNDED PRECEDING
4                           AND CURRENT ROW) AS RESULT1
5
6 FROM WINDOW_TAB1;

```

Result

NO	SAL	RESULT1
1	10	10
2	11	21
3	12	33 10 + 11 + 12
4	12	45 10 + 11 + 12 + 12
5	25	70
6	30	100
7	35	135

☞ SAL 값이 같은 NO 가 3, 4 인 행의 누적합 결과가 서로 다르게 출력됨

### 예제) 연산 범위 응용

SQL1 \*

```

1 SELECT NO, SAL,
2      SUM(SAL) OVER(ORDER BY SAL
3                   ROWS CURRENT ROW) AS RESULT1,
4      SUM(SAL) OVER(ORDER BY SAL
5                   ROWS 1 PRECEDING) AS RESULT2
6
7 FROM WINDOW_TAB1;

```

Result

NO	SAL	RESULT1	RESULT2
1	10	10	10
2	11	11	21
3	12	12	23
4	12	12	24
5	25	25	37
6	30	30	55
7	35	35	65

☞ RESULT1: 현재 행부터 현재 행까지의 합을 계산하기 때문에 각 행의 값을 그대로 반환

RESULT2: 한 개 이전 행부터 현재 행까지의 합을 각 행마다 계산하여 반환

## ● 순위 관련 함수

### 1) RANK(순위)

#### 1-1) RANK WITHIN GROUP

- 특정값에 대한 순위 확인(RANK WITHIN)
- 원도우함수가 아닌 일반함수

#### \*\* 문법

SQL1 \*

```

1 SELECT RANK(값) WITHIN GROUP(ORDER BY 컬럼 ASC|DESC)

```

예제) EMP에서 급여가 3000이면 전체 급여 순위가 얼마?

```
SQL1 *
1 SELECT RANK(3000) WITHIN GROUP(ORDER BY SAL DESC) AS RANK_VALUE
2 FROM EMP;
```

Result

RANK_VALUE
1 2

## 2-2) RANK() OVER()

- 전체 또는 특정 그룹 내 값의 순위 확인
- ORDER BY 절 필수
- 순위를 구할 대상을 ORDER BY 절에 명시(여러 개 나열 가능)
- 그룹 내 순위 구할 시 PARTITION BY 절 사용

\*\* 문법

```
SQL1 *
1 SELECT RANK() OVER([PARTITION BY 컬럼]
2                      ORDER BY 컬럼 ASC|DESC)
```

예제) 각 직원의 급여의 전체 순위(큰 순서대로)

```
SQL1 *
1 SELECT ENAME, DEPTNO, SAL,
2       RANK() OVER(ORDER BY SAL DESC) AS RANK_VALUE1
3   FROM EMP;
```

Result

ENAME	DEPTNO	SAL	RANK_VALUE1
KING	10	5000	1
FORD	20	3000	2
SCOTT	20	3000	2
JONES	20	2975	4
BLAKE	30	2850	5
CLARK	10	2450	6
ALLEN	30	1600	7

이하 생략

예제) 각 직원 이름, 부서번호, 급여, 부서별 급여 순위(큰 순서대로)

```
SQL1 *
1 SELECT ENAME, DEPTNO, SAL,
2      RANK() OVER(PARTITION BY DEPTNO ORDER BY SAL DESC) AS RANK1
3      FROM EMP;
```

Result

	ENAME	DEPTNO	SAL	RANK1
1	KING	10	5000	1
2	CLARK	10	2450	2
3	MILLER	10	1300	3
4	SCOTT	20	3000	1
5	FORD	20	3000	1
6	JONES	20	2975	3
7	ADAMS	20	1100	4
8	SMITH	20	800	5

이하 생략

### 3) DENSE\_RANK

- 누적순위
- 값이 같을 때 동일한 순위 부여 후 다음 순위가 바로 이어지는 순위 부여 방식
- ex) 1 등이 5 명이더라도 그 다음 순위가 2 등

### 4) ROW\_NUMBER

- 연속된 행 번호
- 동일한 순위를 인정하지 않고 단순히 순서대로 나열한대로의 순서 값 리턴

예제) RANK, DENSE\_RANK, ROW\_NUBER 비교

```
SQL1 *
1 SELECT ENAME, DEPTNO, SAL,
2      RANK() OVER(ORDER BY SAL DESC) AS RANK_VALUE1,
3      DENSE_RANK() OVER(ORDER BY SAL DESC) AS RANK_VALUE2,
4      ROW_NUMBER() OVER(ORDER BY SAL DESC) AS RANK_VALUE3
5      FROM EMP;
```

Result

	ENAME	DEPTNO	SAL	RANK_VALUE1	RANK_VALUE2	RANK_VALUE3
1	KING	10	5000	1	1	1
2	FORD	20	3000	2	2	2
3	SCOTT	20	3000	2	2	3
4	JONES	20	2975	4	3	4
5	BLAKE	30	2850	5	4	5
6	CLARK	10	2450	6	5	6
7	ALLEN	30	1600	7	6	7

동순위 처리 차이

후순위 처리 차이

이하 생략

## ● LAG, LEAD

- 행 순서대로 각각 이전 값(LAG), 이후 값(LEAD) 가져오기
- ORDER BY 절 필수

\*\* 문법

```
SQL1 *
1 SELECT LAG(컬럼,
2   [N])
3   OVER ([PARTITION BY 컬럼]
4     ORDER BY 컬럼 [ASC|DESC]);
```

-- 가져올 값을 갖는 컬럼  
-- 몇번째 값을 가져올지(DEFAULT:1)  
-- 행의 이동 그룹  
-- 정렬컬럼

예제) EMP에서 바로 이전 입사자와 급여 비교

```
SQL1 *
1 SELECT ENAME, HIREDATE, SAL,
2   LAG(SAL) OVER(ORDER BY HIREDATE) AS 바로직전상사급여
3   FROM EMP;
```

Result

	ENAME	HIREDATE	SAL	바로직전상사급여
1	SMITH	1980/12/17 00:00:00	800	
2	ALLEN	1981/02/20 00:00:00	1600	800
3	WARD	1981/02/22 00:00:00	1250	1600
4	JONES	1981/04/02 00:00:00	2975	1250

이하 생략

참고) 이전/이후 값 가져올 때 이전 값이 같더라도 항상 행의 순서대로 이전, 이후 하나를 가져옴

따라서 사용자가 이전/이후 값을 가져올 원하는 행 배치를 ORDER BY를 통해 충분히 전달 한 후  
이전/이후 값을 가져오면 됨

```
SQL1 *
1 SELECT EMP.*,
2   LAG(SAL) OVER(ORDER BY DEPTNO, SAL) AS RESULT
3   FROM EMP;
```

Result

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	RESULT
1	7934	MILLER	CLERK	7782	1982/01/23 00:00:00	1300		10	
2	7782	CLARK	MANAGER	7839	1981/06/09 00:00:00	2450		10	1300
3	7839	KING	PRESIDENT		1981/11/17 00:00:00	5000		10	2450
4	7369	SMITH	CLERK	7902	1980/12/17 00:00:00	800		20	5000
5	7876	ADAMS	CLERK	7788	1987/05/23 00:00:00	1100		20	800
6	7566	JONES	MANAGER	7839	1981/04/02 00:00:00	2975		20	1100
7	7788	SCOTT	ANALYST	7566	1987/04/19 00:00:00	3000		20	2975

이하 생략

## ● FIRST\_VALUE, LAST\_VALUE

- 정렬 순서대로 정해진 범위에서의 처음 값, 마지막 값 출력
- 순서와 범위 정의에 따라 최솟값과 최댓값 리턴 가능
- PARTITION BY, ORDER BY 절 생략 가능

**\*\*문법**

```
SQL1 *
1 SELECT FIRST_VALUE(대상) OVER([PARTITION BY 컬럼]
2                               [ORDER BY 컬럼]
3                               [RANGE|ROWS BETWEEN A AND B])
```

**예제) FIRST\_VALUE 를 사용한 최소, 최대 출력**

SQL1 \*

```
1 SELECT ENAME, DEPTNO, SAL,
2       FIRST_VALUE(SAL) OVER(PARTITION BY DEPTNO ORDER BY SAL) AS MIN_VALUE,
3       FIRST_VALUE(SAL) OVER(PARTITION BY DEPTNO ORDER BY SAL DESC) AS MAX_VALUE
4   FROM EMP;
```

Result

	ENAME	DEPTNO	SAL	MIN_VALUE	MAX_VALUE
1	MILLER	10	1300	1300	5000
2	CLARK	10	2450	1300	5000
3	KING	10	5000	1300	5000
4	SMITH	20	800	800	3000
5	ADAMS	20	1100	800	3000
6	JONES	20	2975	800	3000
7	SCOTT	20	3000	800	3000

이하 생략

**예제) LAST\_VALUE 를 사용한 최소, 최대 출력**

SQL1 \*

```
1 SELECT ENAME, DEPTNO, SAL,
2       LAST_VALUE(SAL) OVER(PARTITION BY DEPTNO ORDER BY SAL) AS VALUE1,
3       LAST_VALUE(SAL) OVER(PARTITION BY DEPTNO ORDER BY SAL
4                           RANGE BETWEEN UNBOUNDED PRECEDING
5                           AND UNBOUNDED FOLLOWING) AS MAX_VALUE,
6       LAST_VALUE(SAL) OVER(PARTITION BY DEPTNO ORDER BY SAL DESC
7                           RANGE BETWEEN UNBOUNDED PRECEDING
8                           AND UNBOUNDED FOLLOWING) AS MIN_VALUE
9   FROM EMP
10  WHERE DEPTNO IN (10,20);
```

Result

	ENAME	DEPTNO	SAL	VALUE1	MAX_VALUE	MIN_VALUE
1	MILLER	10	1300	1300	5000	1300
2	CLARK	10	2450	2450	5000	1300
3	KING	10	5000	5000	5000	1300
4	SMITH	20	800	800	3000	800
5	ADAMS	20	1100	1100	3000	800
6	JONES	20	2975	2975	3000	800
7	FORD	20	3000	3000	3000	800
8	SCOTT	20	3000	3000	3000	800

**● NTILE**

- 행을 특정 컬럼 순서에 따라 정해진 수의 그룹으로 나누기 위함 함수
- 그룹 번호가 리턴됨
- ORDER BY 필수

- PARTITION BY 를 사용하여 특정 그룹을 또 원하는 수 만큼 그룹 분리 가능
- 총 행의 수가 명확히 나눠지지 않을 때 앞 그룹의 크기가 더 크게 분리됨  
ex) 14 명 3 개 그룹 분리 시 -> 5, 5, 4 로 나뉨

\*\* 문법

```
SQL1 *
1 SELECT NTILE(N) OVER([PARTITION BY 컬럼]
2                      ORDER BY 컬럼 ASC|DESC)
```

### 예제) NTILE 을 사용한 그룹 분리

```
SQL1 *
1 ▶ SELECT ENAME, SAL, DEPTNO,
2      NTILE(2) OVER (ORDER BY SAL) AS GROUP_NUMBER
3      FROM EMP;
```

Result

	ENAME	SAL	DEPTNO	GROUP_NUMBER
1	SMITH	800	20	1
2	JAMES	950	30	1
3	ADAMS	1100	20	1
4	WARD	1250	30	1
5	MARTIN	1250	30	1
6	MILLER	1300	10	1
7	TURNER	1500	30	1
8	ALLEN	1600	30	2
9	CLARK	2450	10	2
10	BLAKE	2850	30	2
11	JONES	2975	20	2
12	SCOTT	3000	20	2
13	FORD	3000	20	2
14	KING	5000	10	2

### ● 비율관련 함수

#### 1. RATIO\_TO\_REPORT

- 각 값이 전체 합에서 차지하는 비율을 계산
- ORDER BY 절 사용 불가

\*\* 문법

```
SQL1 *
1 RATIO_TO_REPORT(대상) OVER([PARTITION BY 컬럼])
```

### 예제) RATIO\_TO\_REPORT 함수 결과

```
SQL1 *
1 ▶ SELECT SAL,
2      SUM(SAL) OVER() AS TSUM,
3      ROUND(RATIO_TO_REPORT(SAL) OVER(), 4) AS RESULT
4  FROM EMP
5 WHERE DEPTNO = 30
6 ORDER BY SAL;
```

Result

	SAL	TSUM	RESULT
1	950	9400	0.1011
2	1250	9400	0.133
3	1250	9400	0.133
4	1500	9400	0.1596
5	1600	9400	0.1702
6	2850	9400	0.3032

### 2. CUME\_DIST

- 해당 값보다 작거나 같은 값의 비율을 계산
- ORDER BY 절 필수

#### \*\* 문법

```
SQL1 *
1 CUME_DIST() OVER([PARTITION BY 컬럼]
2                      ORDER BY 컬럼 ASC|DESC)
```

### 예) [10, 20, 30, 40, 50] 데이터가 존재할 경우

- CUME\_DIST(10) = 1 / 5 = 0.2
- CUME\_DIST(30) = 3 / 5 = 0.6
- CUME\_DIST(50) = 5 / 5 = 1

### 예제) CUME\_DIST 함수 결과

```
SQL1 *
1 ▶ SELECT SAL,
2      COUNT(SAL) OVER() AS TCOUNT,
3      ROUND(CUME_DIST() OVER(ORDER BY SAL),4) AS RESULT
4  FROM EMP
5 WHERE DEPTNO = 30;
```

Result

	SAL	TCOUNT	RESULT
1	950	6	0.1667 1/6
2	1250	6	0.5 3/6
3	1250	6	0.5
4	1500	6	0.6667
5	1600	6	0.8333
6	2850	6	1

☞ 각 행마다 각 값보다 작거나 같은 값들의 비율을 출력하기 때문에 값이 같은 1250의 경우 결과값이 같다

### 3. PERCENT\_RANK

- 주어진 값이 데이터 집합 내에서 상대적으로 어디에 위치하는지를 비율(퍼센트)로 나타내는 함수
- 0과 1 사이의 값으로 표현
- ORDER BY 절 필수

\*\* 문법

```
SQL1 *
1 PERCENT_RANK() OVER ([PARTITION BY 컬럼]
2           ORDER BY 컬럼 ASC|DESC)
```

$$\text{PERCENT\_RANK} = \frac{\text{RANK} - 1}{(\text{총 행 수}) - 1}$$

예) [10, 20, 30, 40, 50] 데이터가 존재할 경우

- PERCENT\_RANK(10) = (1 - 1) / (5 - 1) = 0
- PERCENT\_RANK(30) = (3 - 1) / (5 - 1) = 0.5
- PERCENT\_RANK(50) = (5 - 1) / (5 - 1) = 1

예제) PERCENT\_RANK 결과

```
SQL1 *
1 SELECT SAL,
2       COUNT(SAL) OVER() AS TCOUNT,
3       ROUND(PERCENT_RANK() OVER(ORDER BY SAL), 4) AS RESULT
4   FROM EMP
5 WHERE DEPTNO = 30;
```

Result

	SAL	TCOUNT	RESULT
1	950	6	0
2	1250	6	0.2
3	1250	6	0.2
4	1500	6	0.6
5	1600	6	0.8
6	2850	6	1

☞ 총 6 개의 행이므로 1 / (6 - 1) 만큼의 누적 비율을 반환함

값이 같은 경우는 동순위를 가지므로 PERCENT\_RANK 결과가 같다.

## 2-13. TOP N QUERY

### ● TOP N QUERY

- 페이지 처리를 효과적으로 수행하기 위해 사용
- 전체 결과에서 특정 N 개 추출

예) 성적 상위자 3 명

## ● TOP-N 행 추출 방법

1. ROWNUM
2. RANK
3. FETCH
4. TOP N(SQL Server)

## ● ROWNUM

- 출력된 데이터 기준으로 행 번호 부여
- 절대적인 행 번호가 아닌 가상의 번호이므로 특정 행을 지정할 수 없음(=연산 불가)
- 첫 번째 행이 증가한 이후 할당되므로 '>' 연산 사용 불가(0은 가능)

### 예제) ROWNUM 을 출력 형태

SQL1 \*

```
1 ▶ SELECT ROWNUM, EMP.*  
2   FROM EMP  
3 WHERE SAL >= 1500;
```

Result

	ROWNUM	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1	7499	ALLEN	SALESMAN	7698	1981/02/20 00:00:00	1600	300	30
2	2	7566	JONES	MANAGER	7839	1981/04/02 00:00:00	2975		20
3	3	7698	BLAKE	MANAGER	7839	1981/05/01 00:00:00	2850		30
4	4	7782	CLARK	MANAGER	7839	1981/06/09 00:00:00	2450		10
5	5	7788	SCOTT	ANALYST	7566	1987/04/19 00:00:00	3000		20
6	6	7839	KING	PRESIDENT		1981/11/17 00:00:00	5000		10
7	7	7844	TURNER	SALESMAN	7698	1981/09/08 00:00:00	1500	0	30
8	8	7902	FORD	ANALYST	7566	1981/12/03 00:00:00	3000		20

### 예제) ROWNUM 잘못된 사용 1

SQL1 \*

```
1 ▶ SELECT *  
2   FROM EMP  
3 WHERE ROWNUM > 1;
```

Result

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO

☞ 크다 조건 전달 불가

### 예제) ROWNUM 잘못된 사용 2

```
SQL1 *
1 > SELECT *
2   FROM EMP
3 WHERE ROWNUM = 4;
```

Result

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
-------	-------	-----	-----	----------	-----	------	--------

☞ 항상 불변하는 절대적 번호가 아니므로 '=' 연산자 단독 전달 불가

### 예제) ROWNUM 올바른 사용

```
SQL1 *
1 > SELECT EMPNO, ENAME, DEPTNO, SAL
2   FROM EMP
3 WHERE ROWNUM <= 5;
```

Result

EMPNO	ENAME	DEPTNO	SAL
1 7369	SMITH	20	800
2 7499	ALLEN	30	1600
3 7521	WARD	30	1250
4 7566	JONES	20	2975
5 7654	MARTIN	30	1250

☞ EQUAL 비교 시 작다(<)와 함께 사용하면 1부터 순서대로 뽑을 수 있기 때문에 출력 가능함  
 ☞ 정렬 순서에 따라 출력되는 ROWNUM이 달라짐

### 예제) EMP 테이블에서 급여가 높은 순서대로 상위 5명의 직원 정보 출력

\*\* 잘못된 예

```
SQL1 *
1 > SELECT EMPNO, ENAME, DEPTNO, SAL
2   FROM EMP
3 WHERE ROWNUM <= 5
4 ORDER BY SAL DESC;
```

Result

EMPNO	ENAME	DEPTNO	SAL
1 7566	JONES	20	2975
2 7499	ALLEN	30	1600
3 7654	MARTIN	30	1250
4 7521	WARD	30	1250
5 7369	SMITH	20	800

☞ 실제로 상위 5명 출력 안됨(급여최대가 5000임)  
 ☞ 추출 원리 : WHERE 절에 의해 먼저 5개를 추출 뒤 이 결과 집합에 대해 정렬 수행

**\*\* 해결 : 먼저 서브쿼리를 사용하여(인라인뷰) SAL에 대해 내림차순 정렬을 해놓고 상위 5개 가져옴**

```
SQL1 *
1 ▷ SELECT *
2   FROM (SELECT *
3          FROM EMP
4         ORDER BY SAL DESC)
5   WHERE ROWNUM <= 5
6   ORDER BY SAL DESC;
```

Result

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
1	7839	KING	PRESIDENT		1981/11/17 00:00:00	5000
2	7788	SCOTT	ANALYST	7566	1987/04/19 00:00:00	3000
3	7902	FORD	ANALYST	7566	1981/12/03 00:00:00	3000
4	7566	JONES	MANAGER	7839	1981/04/02 00:00:00	2975
5	7698	BLAKE	MANAGER	7839	1981/05/01 00:00:00	2850

☞ 즉 ROWNUM이 결정되기 전에 먼저 데이터 정렬순서를 바꿔놓는 방법

예제) EMP 테이블에서 급여가 높은 순서대로 4 ~ 6 번째 해당하는 직원 정보 출력

**\*\* 잘못된 예**

```
SQL1 *
1 ▷ SELECT *
2   FROM (SELECT *
3          FROM EMP
4         ORDER BY SAL DESC)
5   WHERE ROWNUM BETWEEN 4 AND 6
6   ORDER BY SAL DESC;
```

Result

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO

☞ ROWNUM 시작 값(1)이 정의되지 않았으므로 1을 건너뛰고 그 다음 행 번호에 대한 추출 불가

**\*\* 해결 : 인라인뷰에서 각 행마다의 순위를 직접 부여**

```
SQL1 *
1 ▷ SELECT *
2   FROM (SELECT ROWNUM AS RN, A.*
3          FROM (SELECT *
4                  FROM EMP
5                 ORDER BY SAL DESC) A) B
6   WHERE RN BETWEEN 4 AND 6
7   ORDER BY SAL DESC;
```

Result

	RN	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
1	4	7566	JONES	MANAGER	7839	1981/04/02 00:00:00	2975
2	5	7698	BLAKE	MANAGER	7839	1981/05/01 00:00:00	2850
3	6	7782	CLARK	MANAGER	7839	1981/06/09 00:00:00	2450

☞ 서브쿼리를 통해 얻은 결과에 ROWNUM을 다시 부여하여 새로운 테이블인 것처럼 사용(인라인뷰)

\*\* 해결 : 원도우 함수의 RANK 사용

```

SQL1 *
1 SELECT *
2   FROM (SELECT EMP.*,
3                 RANK() OVER(ORDER BY SAL DESC) AS RN
4            FROM EMP) A
5   WHERE RN BETWEEN 4 AND 6
6 ORDER BY SAL DESC;

```

Result

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	RN
1	7566	JONES	MANAGER	7839	1981/04/02 00:00:00	2975		20	4
2	7698	BLAKE	MANAGER	7839	1981/05/01 00:00:00	2850		30	5
3	7782	CLARK	MANAGER	7839	1981/06/09 00:00:00	2450		10	6

● FETCH 절

- 출력될 행의 수를 제한하는 절
- ORACLE 12C 이상부터 제공(이전버전에는 ROWNUM 주로 사용)
- SQL Server 사용 가능
- ORDER BY 절 뒤에 사용(내부 파싱 순서도 ORDER BY 뒤)

\*\* 문법

```

SQL1 *
1 SELECT
2   FROM
3   WHERE
4   GROUP BY
5   HAVING
6   ORDER BY
7   OFFSET N { ROW | ROWS }
8   FETCH { FIRST | NEXT } N { ROW | ROWS } ONLY;

```

- **OFFSET** : 건너뛸 행의 수

ex) 성적 높은 순 1등 제외, 나머지 3명

- **N** : 출력할 행의 수

- **FETCH** : 출력할 행의 수를 전달하는 구문

- **FIRST** : OFFSET을 쓰지 않았을 때 처음부터 N 행 출력 명령

- **NEXT** : OFFSET을 사용했을 경우 제외한 행 다음부터 N 행 출력 명령

- **ROW | ROWS** : 행의 수에 따라 하나일 경우 단수, 여러 값이면 복수형(특별히 구분하지 않아도 됨)

예제) EMP에서 SAL 순서대로 상위 5명(19C에서 실행)

SQL1 \*

```

1 SELECT EMPNO, ENAME, JOB, SAL
2   FROM EMP
3 ORDER BY SAL DESC FETCH FIRST 5 ROWS ONLY;

```

Result

	EMPNO	ENAME	JOB	SAL
1	7839	KING	PRESIDENT	5000
2	7788	SCOTT	ANALYST	3000
3	7902	FORD	ANALYST	3000
4	7566	JONES	MANAGER	2975
5	7698	BLAKE	MANAGER	2850

예제) EMP 테이블에서 급여가 높은 순서대로 4 ~ 5 번째 해당하는 직원 정보 출력

SQL1 \*

```

1 SELECT EMPNO, ENAME, JOB, SAL
2   FROM EMP
3 ORDER BY SAL DESC
4 OFFSET 3 ROW
5 FETCH FIRST 2 ROW ONLY;

```

Result

	EMPNO	ENAME	JOB	SAL
1	7566	JONES	MANAGER	2975
2	7698	BLAKE	MANAGER	2850

### ● TOP N 쿼리

- SQL Server에서의 상위 n 개 행 추출 문법
- 서브쿼리 사용 없이 하나의 쿼리로 정렬된 순서대로 상위 n 개 추출 가능
- WITH TIES를 사용하여 동순위까지 함께 출력 가능

\*\* 문법

SQL1 \*

```

1 SELECT TOP N 컬럼1, 컬럼2, ...
2   FROM 테이블명
3 ORDER BY 정렬컬럼명 [ASC|DESC] ...

```

예제) EMP 테이블의 상위 급여자 2명 출력(SQL Server에서 수행)

SQL1 \*

```

1 SELECT TOP 2 ENAME, SAL
2   FROM EMP
3 ORDER BY SAL DESC;

```

Result

	ENAME	SAL
1	KING	5000
2	SCOTT	3000

SQL1 \*

```

1 SELECT TOP 2 WITH TIES ENAME, SAL
2 FROM EMP
3 ORDER BY SAL DESC;

```

Result

	ENAME	SAL
1	KING	5000
2	SCOTT	3000
3	FORD	3000

SAL은 큰 순서대로 5000, 3000, 3000 이라 3000이 공동 2위이지만, TOP 2는 2개만 출력, WITH TIES를 사용하면 동순위 행도 함께 출력 가능

## 2-14. 계층형 질의

### ● 계층형 질의

- 하나의 테이블 내 각 행끼리 관계를 가질 때, 연결고리를 통해 행과 행 사이의 계층(depth)을 표현하는 기법
- ex) DEPT2에서의 부서별 상하관계
- PRIOR의 위치에 따라 연결하는 데이터가 달라짐

### \*\* 문법

SQL1 \*

```

1 SELECT 컬럼명
2   FROM 테이블명
3   START WITH 시작조건      -- 시작점을 지정하는 조건 전달
4 CONNECT BY [NOCYCLE] PRIOR 연결조건; -- 시작점 기준으로 연결 데이터를 찾아가는 조건

```

\*\* START WITH : 데이터를 출력할 시작 지정하는 조건

\*\* CONNECT BY PRIOR : 행을 이어나갈 조건

\*\* NOCYCLE : 순환이 발생하면 무한 루프가 될 수 있기 때문에 이를 방지하고자 사용

예제) DEPT2 테이블에 대해 각 부서의 레벨을 출력(최상위 부서가 1 레벨)

**\*\* 올바른 예**

SQL1 \*

```

1 SELECT D.* , LEVEL
2   FROM DEPT2 D
3  START WITH PDEPT IS NULL
4 CONNECT BY PRIOR DCODE = PDEPT;

```

Result

	DCODE	DNAME	PDEPT	AREA	LEVEL
1	0001	사장실		포항본사	1
2	1000	경영지원부	0001	서울지사	2
3	1001	재무관리팀	1000	서울지사	3
4	1002	총무팀	1000	서울지사	3
5	1003	기술부	0001	포항본사	2
6	1004	H/W지원	1003	대전지사	3
7	1005	S/W지원	1003	경기지사	3
8	1006	영업부	0001	포항본사	2
9	1007	영업기획팀	1006	포항본사	3
10	1008	영업1팀	1007	부산지사	4
11	1009	영업2팀	1007	경기지사	4
12	1010	영업3팀	1007	서울지사	4
13	1011	영업4팀	1007	울산지사	4

☞ 사장실의 DCODE를 넘겨 다시 각 행들의 PDEPT와 비교해야 하므로 먼저 정해져야 하는 값의 방향에 PRIOR 전달!

**\*\* 잘못된 예**

SQL1 \*

```

1 SELECT D.* , LEVEL
2   FROM DEPT2 D
3  START WITH PDEPT IS NULL
4 CONNECT BY DCODE = PRIOR PDEPT;

```

Result

	DCODE	DNAME	PDEPT	AREA	LEVEL
1	0001	사장실		포항본사	1

## ● 계층형 질의 조건

### 1. CONNECT BY 절 조건

- 계층 구조를 설정하는 조건
- START WITH 절에서 시작한 데이터로부터 부모-자식 관계를 탐색하는 규칙을 정의
- 이 조건이 성립하지 않으면, 더 이상 자식 관계를 연결하지 않음
- START WITH 절 데이터가 항상 출력됨

## 2. WHERE 절 조건

- 전체 결과를 필터링하는 데 사용
- START WITH 와 CONNECT BY 로 연결된 모든 결과가 출력된 후, WHERE 절 조건에 맞는 데이터만 선택하여 출력
- START WITH 절 데이터가 조건에 맞지 않는 경우 생략됨

### 예제) CONNECT BY 절 조건 사용

```
SQL1 *
1 SELECT *
2   FROM DEPT2
3  START WITH PDEPT IS NULL
4 CONNECT BY PRIOR DCODE = PDEPT
5      AND AREA = '서울지사';

Result
Grid Result Server Output Text Output Explain Plan Statistics
DCODE DNAME PDEPT AREA
1 0001 사장실 포항본사
2 1000 경영지원부 0001 서울지사
3 1001 재무관리팀 1000 서울지사
4 1002 총무팀 1000 서울지사
```

DCODE	DNAME	PDEPT	AREA
0001	사장실		포항본사
1000	경영지원부	0001	서울지사
1001	재무관리팀	1000	서울지사
1002	총무팀	1000	서울지사
1003	기술부	0001	포항본사
1004	H/W지원	1003	대전지사
1005	S/W지원	1003	경기지사
1006	영업부	0001	포항본사
1007	영업기획팀	1006	포항본사
1008	영업1팀	1007	부산지사
1009	영업2팀	1007	경기지사
1010	영업3팀	1007	서울지사
1011	영업4팀	1007	울산지사

### 예제) WHERE 절 조건 사용

```
SQL1 *
1 SELECT D.* , LEVEL
2   FROM DEPT2 D
3  WHERE AREA = '서울지사'
4  START WITH PDEPT IS NULL
5 CONNECT BY PRIOR DCODE = PDEPT;

Result
Grid Result Server Output Text Output Explain Plan Statistics
DCODE DNAME PDEPT AREA LEVEL
1 1000 경영지원부 0001 서울지사 2
2 1001 재무관리팀 1000 서울지사 3
3 1002 총무팀 1000 서울지사 3
4 1010 영업3팀 1007 서울지사 4
```

DCODE	DNAME	PDEPT	AREA	LEVEL
0001	사장실		포항본사	1
1000	경영지원부	0001	서울지사	2
1001	재무관리팀	1000	서울지사	3
1002	총무팀	1000	서울지사	3
1003	기술부	0001	포항본사	2
1004	H/W지원	1003	대전지사	3
1005	S/W지원	1003	경기지사	3
1006	영업부	0001	포항본사	2
1007	영업기획팀	1006	포항본사	3
1008	영업1팀	1007	부산지사	4
1009	영업2팀	1007	경기지사	4
1010	영업3팀	1007	서울지사	4
1011	영업4팀	1007	울산지사	4

☞ WHERE 절 없이 CONNECT BY 로 부모-자식 관계를 아래와 같이 모두 연결한 뒤  
WHERE 절에 성립하는 데이터만 선택하여 최종 출력됨

### \* 계층형 질의 가상 컬럼

- 1) LEVEL : 각 DEPTH 를 표현(시작점부터 1)
- 2) CONNECT\_BY\_ISLEAF : LEAF NODE(최하위노드) 여부(참:1, 거짓:0)

### \* 계층형 질의 가상 함수

- 1) CONNECT\_BY\_ROOT 컬럼명 : 루트노드의 해당하는 컬럼값
- 2) SYS\_CONNECT\_BY\_PATH(컬럼, 구분자) : 이어지는 경로 출력
- 3) ORDER SIBLINGS BY 컬럼 : 같은 LEVEL 일 경우 정렬 수행
- 4) CONNECT\_BY\_ISCYCLE : 계층형 쿼리의 결과에서 순환이 발생했는지 여부

### 예제) 계층형 질의절 가상 컬럼 및 함수의 사용

SQL1 \*

```

1 SELECT D.* , LEVEL,
2       CONNECT_BY_ROOT DNAME ,
3       SYS_CONNECT_BY_PATH(DNAME , '-' )
4   FROM DEPARTMENT D
5  START WITH DEPTNO = 10
6 CONNECT BY PRIOR DEPTNO = PDEPT
7 ORDER SIBLINGS BY DNAME ;

```

Result

	DEPTNO	DNAME	PDEPT	BUILD	LEVEL	CONNECT_BY_ROOTDNAME	SYS_CONNECT_BY_PATH(DNAME, '-')
1	10	공과대학			1	공과대학	-공과대학
2	200	자연과학부		10	2	공과대학	-공과대학-자연과학부
3	201	수학과		200	3	공과대학	-공과대학-자연과학부-수학과
4	202	통계학과		200	3	공과대학	-공과대학-자연과학부-통계학과
5	203	화학공학과		200	3	공과대학	-공과대학-자연과학부-화학공학과
6	100	컴퓨터공학부		10	2	공과대학	-공과대학-컴퓨터공학부
7	102	빅데이터융합과		100	3	공과대학	-공과대학-컴퓨터공학부-빅데이터융합과
8	103	소프트웨어공학과		100	3	공과대학	-공과대학-컴퓨터공학부-소프트웨어공학과
9	101	컴퓨터공학과		100	3	공과대학	-공과대학-컴퓨터공학부-컴퓨터공학과

☞ ORDER SIBLINGS BY 를 사용하여 같은 레벨일 경우 DNAME 오름차순으로 정렬,  
 2 레벨은 자연과학부 < 컴퓨터공학부 순서대로 출력되며,  
 자연과학부 내 3 레벨은 수학과 < 통계학과 < 화학공학과 순서대로 리턴되었음

### 예제) NOCYCLE 옵션

#### < EMPLOYEES DATA >

	EMPLOYEE_ID	DEPARTMENT_ID	MANAGER_ID	NAME	HIRE_DATE
1	1000		2000	홍은혜	2024/06/24 00:10:21
2	2000		1000	박승민	2024/06/24 00:10:22

#### < NOCYCLE 옵션 없이 - ERROR 발생 >

SQL1 \*

```

1 SELECT EMPLOYEE_ID , NAME , LEVEL
2   FROM EMPLOYEES2
3  START WITH EMPLOYEE_ID = 1000
4 CONNECT BY PRIOR EMPLOYEE_ID = MANAGER_ID ;

```

Result

EMPLOYEE_ID	NAME	LEVEL
-------------	------	-------

ORA-01436: CONNECT BY의 루프가 발생되었습니다 | Ln 4, Col 43

☞ 1000 번 직원의 매니저는 2000 번 사원인데, 2000 번 사원도 1000 번 직원이 매니저이므로 서로 순환구조를 가짐. 이런 관계에서 NOCYCLE 없이는 에러가 발생함

## &lt; NOCYCLE 옵션 수행 시 - 정상 출력 &gt;

The screenshot shows the SQL1\* tab with the following SQL code:

```

1 SELECT EMPLOYEE_ID, NAME, LEVEL,
2       CONNECT_BY_ISCYCLE AS IS_CYCLE
3   FROM EMPLOYEES2
4  START WITH EMPLOYEE_ID = 1000
5 CONNECT BY NOCYCLE PRIOR EMPLOYEE_ID = MANAGER_ID;

```

The Result tab displays the output:

	EMPLOYEE_ID	NAME	LEVEL	IS_CYCLE
1	1000	홍은혜	1	0
2	2000	박승민	2	1

## 2-15. PIVOT 과 UNPIVOT

(데이터의 구조를 변경하는 기능)

## ● 데이터의 구조

## 1) LONG DATA(Tidy data)

- 하나의 속성이 하나의 컬럼으로 정의되어 값들이 여러 행으로 쌓이는 구조
- RDBMS 의 테이블 설계 방식
- 다른 테이블과의 조인 연산이 가능한 구조

## \*\* LONG DATA

	EMPNO	ENAME	JOB	MGR	HIREDATE
1	7369	SMITH	CLERK	7902	1980/12/17 00:00:00
2	7499	ALLEN	SALESMAN	7698	1981/02/20 00:00:00
3	7521	WARD	SALESMAN	7698	1981/02/22 00:00:00
4	7566	JONES	MANAGER	7839	1981/04/02 00:00:00
5	7654	MARTIN	SALESMAN	7698	1981/09/28 00:00:00
6	7698	BLAKE	MANAGER	7839	1981/05/01 00:00:00
7	7782	CLARK	MANAGER	7839	1981/06/09 00:00:00
8	7788	SCOTT	ANALYST	7566	1987/04/19 00:00:00
9	7839	KING	PRESIDENT		1981/11/17 00:00:00

## 2) WIDE DATA(Cross table)

- 행과 컬럼에 유의미한 정보 전달을 목적으로 작성하는 교차표
- 하나의 속성값이 여러 컬럼으로 분리되어 표현
- RDBMS 에서 WIDE 형식으로 테이블 설계 시 값이 추가될 때 마다 컬럼이 추가돼야 하므로 비효율적!
- 다른 테이블과의 조인 연산이 불가함
- 주로 데이터를 요약할 목적으로 사용

## \*\* WIDE DATA

	JOB	10	20	30
1	CLERK	1	2	1
2	SALESMAN	0	0	4
3	PRESIDENT	1	0	0
4	MANAGER	1	1	1
5	ANALYST	0	2	0

☞ 컬럼의 정보는 부서번호로, 하나의 관찰대상(속성)을 한 컬럼으로 정의하지 않고 값의 종류별로 컬럼을 분리하였음

## ● 데이터 구조 변경

### 1) PIVOT : LONG -> WIDE

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980/12/17 00:00:00	800		20
2	7499	ALLEN	SALESMAN	7698	1981/02/20 00:00:00	1600	300	30
3	7521	WARD	SALESMAN	7698	1981/02/22 00:00:00	1250	500	30
4	7566	JONES	MANAGER	7839	1981/04/02 00:00:00	2975		20
5	7654	MARTIN	SALESMAN	7698	1981/09/28 00:00:00	1250	1400	30
6	7698	BLAKE	MANAGER	7839	1981/05/01 00:00:00	2850		30
7	7782	CLARK	MANAGER	7839	1981/06/09 00:00:00	2450		10
8	7788	SCOTT	ANALYST	7566	1987/04/19 00:00:00	3000		20
9	7839	KING	PRESIDENT		1981/11/17 00:00:00	5000		10
10	7844	TURNER	SALESMAN	7698	1981/09/08 00:00:00	1500	0	30
11	7876	ADAMS	CLERK	7788	1987/05/23 00:00:00	1100		20
12	7900	JAMES	CLERK	7698	1981/12/03 00:00:00	950		30
13	7902	FORD	ANALYST	7566	1981/12/03 00:00:00	3000		20
14	7934	MILLER	CLERK	7782	1982/01/23 00:00:00	1300		10

JOB	10	20	30
1 CLERK	1	2	1
2 SALESMAN	0	0	4
3 PRESIDENT	1	0	0
4 MANAGER	1	1	1
5 ANALYST	0	2	0

### 2) UNPIVOT : WIDE -> LONG

	MONTH	월	화	수	목	금	토	일
1	01	10	20	30	40	45	35	60
2	02	39	45	75	34	65	45	33
3	03	13	24	67	43	45	34	56
4	04	34	50	34	22	24	43	44

	MONTH	요일	구매건수
1	01	월	10
2	01	화	20
3	01	수	30
4	01	목	40
5	01	금	45
6	01	토	35
7	01	일	60
8	02	월	39
9	02	화	45
10	02	수	75
11	02	목	34
12	02	금	65
13	02	토	45
14	02	일	33
15	03	월	13
16	03	화	24
17	03	수	67
18	03	목	43
19	03	금	45
20	03	토	34
21	03	일	56

## ● PIVOT

- 교차표를 만드는 기능
- STACK 컬럼, UNSTACK 컬럼, VALUE 컬럼의 정의가 중요!
- FROM 절에 STACK, UNSTACK, VALUE 컬럼명만 정의 필요(필요 시 서브쿼리 사용하여 필요 컬럼 제한)
- PIVOT 절에 UNSTACK, VALUE 컬럼명 정의
- PIVOT 절 IN 연산자에 UNSTACK 컬럼 값은 정의
- FROM 절에 선언된 컬럼 중 PIVOT 절에서 선언한 VALUE 컬럼, UNSTACK 컬럼을 제외한 모든 컬럼은 STACK 컬럼이 됨

UNSTACK 컬럼							
MONTH	월	화	수	목	금	토	일
1 01	10	20	30	40	45	35	60
2 02	39	45	75	34	65	45	33
3 03	13	24	67	43	45	34	56
4 04	34	50	34	22	24	43	44

STACK 컬럼                    VALUE 컬럼

### \*\* 문법

```
SQL1 *
1 SELECT *
2   FROM 테이블명 또는 서브쿼리
3   PIVOT (VALUE컬럼명 FOR UNSTACK컬럼명 IN (값1, 값2, 값3));
```

\* 반드시 FROM 절에 STACK 컬럼, UNSTACK 컬럼, VALUE 컬럼 모두 명시!

예제) EMP 테이블에서 아래와 같이 JOB 별 DEPTNO 별 도수(COUNT) 출력

JOB	10	20	30
1 CLERK	1	2	1
2 SALESMAN	0	0	4
3 PRESIDENT	1	0	0
4 MANAGER	1	1	1
5 ANALYST	0	2	0

### < 정답 >

```
SQL1 *
1 ▶ SELECT *
2   FROM (SELECT EMPNO, JOB, DEPTNO FROM EMP)
3   PIVOT (COUNT(EMPNO) FOR DEPTNO IN (10,20,30));
```

### Result

JOB	10	20	30
1 CLERK	1	2	1
2 SALESMAN	0	0	4
3 PRESIDENT	1	0	0
4 MANAGER	1	1	1
5 ANALYST	0	2	0

\*주의 : 이 때 FROM 절 서브쿼리 안에 JOB이 없으면 아래와 같이 그냥 부서별로의 도수가 출력됨

```
SQL1 *
1 SELECT *
2   FROM (SELECT EMPNO, DEPTNO FROM EMP)
3   PIVOT (COUNT(EMPNO) FOR DEPTNO IN (10,20,30));
```

Result

	10	20	30
1	3	5	6

\*주의 : FROM 절에 서브쿼리로 컬럼을 제한하지 않으면 STACK 컬럼이 많아짐!!

```
SQL1 *
1 SELECT *
2   FROM EMP
3   PIVOT (COUNT(EMPNO) FOR DEPTNO IN (10, 20, 30));
```

Result

	ENAME	JOB	MGR	HIREDATE	SAL	COMM	10	20	30
1	JAMES	CLERK	7698	1981/12/03 00:00:00	950		0	0	1
2	MILLER	CLERK	7782	1982/01/23 00:00:00	1300		1	0	0
3	SCOTT	ANALYST	7566	1987/04/19 00:00:00	3000		0	1	0
4	ADAMS	CLERK	7788	1987/05/23 00:00:00	1100		0	1	0
5	SMITH	CLERK	7902	1980/12/17 00:00:00	800		0	1	0
6	CLARK	MANAGER	7839	1981/06/09 00:00:00	2450		1	0	0

이하 생략

☞ FROM 절에 서브쿼리로 필요한 컬럼만 정의하지 않으면 EMP 테이블의 모든 컬럼 중 PIVOT 절에 선언된 EMPNO, DEPTNO 컬럼을 제외한 모든 컬럼이 STACK 처리 됨

예제) 다음의 테이블에서 성별, 연도별 구매량 총합을 표현하는 교차표 작성

```
SQL1 *
1 SELECT * FROM UNSTACK_TEST;
```

Result

	년도	성별	지역	구매량
1	2008	남자	서울	1
2	2008	남자	경기	2
3	2008	여자	서울	3
4	2008	여자	경기	4
5	2009	남자	서울	5
6	2009	남자	경기	6
7	2009	여자	서울	7
8	2009	여자	경기	8



	성별	2008	2009
1	남자	3	11
2	여자	7	15

## &lt; 정답 &gt;

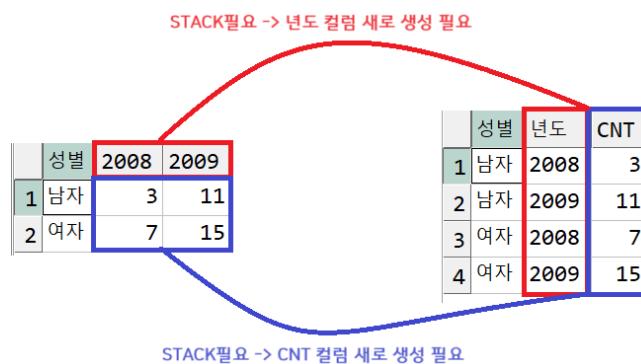
```
SQL1 *
1 SELECT *
2   FROM (SELECT 년도, 성별, 구매량 FROM UNSTACK_TEST)
3   PIVOT (SUM(구매량) FOR 년도 IN (2008,2009));
```

Result

성별	2008	2009
1 남자	3	11
2 여자	7	15

## ● UNPIVOT

- WIDE 데이터를 LONG 데이터로 변경하는 문법
- STACK 컬럼 : 이미 UNSTACK 되어 있는 여러 컬럼을 하나의 컬럼으로 STACK 시 새로 만들 컬럼이름 (사용자 정의)
- VALUE 컬럼 : 교차표에서 셀 자리(VALUE)값을 하나의 컬럼으로 표현하고자 할 때 새로 만들 컬럼명 (사용자 정의)
- 값 1, 값 2... : 실제 UNSTACK 되어 있는 컬럼이름들



## \*\* 문법

```
SQL1 *
1 SELECT *
2   FROM 테이블명 또는 서브쿼리
3 UNPIVOT (VALUE컬럼명 FOR STACK컬럼명 IN (값1, 값2, ...));
```

예제) 위 UNSTACK\_TEST PIVOT 결과가 STACK\_TEST 테이블에 저장되어 있을 때, 다시 STACK\_TEST 테이블의 값을 UNSTACK\_TEST 형태로 변경(STACK 처리)

## &lt; 정답 &gt;

```
SQL1 *
1 SELECT *
2   FROM STACK_TEST
3 UNPIVOT (CNT FOR 년도 IN ("2008", "2009"));
```

Result

성별	년도	CNT
1 남자	2008	3
2 남자	2009	11
3 여자	2008	7
4 여자	2009	15

☞ IN 뒤에 값은 UNSTACK 데이터의 컬럼명이 숫자이지만 컬럼명은 문자로 저장되므로 쌍따옴표 전달 필요!

예제) 아래 테이블 STACK 처리(LONG DATA 로 변환)

The screenshot shows a SQL query in the SQL1 window:

```
1 SELECT *
2 FROM TT5;
```

The result set is displayed in a grid format under the 'Grid Result' tab. The columns are labeled 'MONTH' and the days of the week: 월, 화, 수, 목, 금, 토, 일. The data is as follows:

MONTH	월	화	수	목	금	토	일
1 01	10	20	30	40	45	35	60
2 02	39	45	75	34	65	45	33
3 03	13	24	67	43	45	34	56
4 04	34	50	34	22	24	43	44

< 정답 >

The screenshot shows a modified SQL query in the SQL1 window:

```
1 SELECT *
2 FROM TT5
3 UNPIVOT (구매건수 FOR 요일 IN (월, 화, 수, 목, 금, 토, 일));
```

The result set is displayed in a grid format under the 'Grid Result' tab. The columns are labeled 'MONTH' and '요일'. The data is as follows:

	MONTH	요일	구매건수
1	01	월	10
2	01	화	20
3	01	수	30
4	01	목	40
5	01	금	45
6	01	토	35
7	01	일	60
8	02	월	39
9	02	화	45
10	02	수	75
11	02	목	34

[이하 생략]

☞ 월, 화, 수, 목, .... 값들은 컬럼명이므로 컬럼명과 테이블명처럼 대명사(객체이름)는 쌍따옴표를 붙이지 않음  
주의!

## 2-16. 정규 표현식

### ● 정규 표현식

- 문자열의 공통된 규칙을 보다 일반화 하여 표현하는 방법
- 정규 표현식 사용 가능한 문자함수 제공(regexp\_replace, regexp\_substr, regexp\_instr, ...)  
 ex) 숫자를 포함하는, 숫자로 시작하는 4 자리, 두번째 자리가 A 인 5 글자

## 예제) 일반화 규칙 찾아내기



## \* 정규 표현식 종류

					그룹번호
\d	Digit, 0, 1, 2, ..., 9	[ab]	a 또는 b의 한 글자	\n	
\D	숫자가 아닌 것	[^ab]	a 와 b 제외한 모든 문자	[:alnum:]	문자와 숫자
\s	공백	[0-9]	숫자	[:alpha:]	문자
\S	공백이 아닌 것	[A-Z]	영어 대문자	[:blank:]	공백
\w	단어	[a-z]	영어 소문자	[:cntrl:]	제어문자
\W	단어가 아닌 것	[A-z]	모든 영문자	[:digit:]	digits
\t	Tab	i+	i가 1회 이상 반복	[:graph:]	graphical characters [:alnum:], [:punct:]
\n	New line(엔터문자)	i*	i가 0회 이상 반복	[:lower:]	소문자
^	시작되는글자	i?	0회에서 1회 반복	[:print:]	숫자, 문자, 특수문자, 공백 모두
\$	마지막글자	i{n}	i가 n회 반복	[:punct:]	특수문자
\	Escape character (뒤에 기호 의미 제거)	i{n1,n2}	i가 n1에서 n2회 반복	[:space:]	공백문자
	또는	i{n,}	i가 n회 이상 반복	[:upper:]	대문자
.	엔터를 제외한 모든 한 글자	0	그룹지정	[:xdigit:]	16진수

## 예제) 전화번호의 일반화

글자	1차 일반화 결과	2차 일반화 결과
tel)02-999-4456	tel\)[0-9-]+	
tel02-999-4456	tel[0-9-]+	tel\)?[0-9-]+

- ☞ 전화번호는 숫자와 -으로 구성 -> [0-9-]+ 로 표현 가능([] 안에 들어가는 패턴이 한자리의 문자열을 구성할 수 있는 값들)
- ☞ 두 전화번호가 tel 값은 동시에 있지만, )가 있는 경우와 없는 경우를 모두 표현 -> ?사용(?는 값이 없거나 1 개 있음을 의미)

## ● REGEXP\_REPLACE

- 정규식 표현을 사용한 문자열 치환 가능

### \*\* 문법

(대상, 찾을문자열, [바꿀문자열], [검색위치], [발견횟수], [옵션])

### 1. 특징

- 바꿀문자열 생략 시 문자열 삭제
- 검색위치 생략 시 1
- 발견횟수 생략 시 0(모든)

### 2. 옵션

- c : 대소를 구분하여 검색
- i : 대소를 구분하지 않고 검색
- m : 패턴을 다중라인으로 선언 가능

### 예제) ID에서 숫자 삭제

```
SQL1 *
1 SELECT ID,
2      REGEXP_REPLACE(ID, '\d', '') AS RESULT1,
3      REGEXP_REPLACE(ID, '[:digit:]', '') AS RESULT2
4 FROM PROFESSOR;
```

Result

	ID	RESULT1	RESULT2
1	loveu	loveu	loveu
2	a1234	a	a
3	putty	putty	putty
4	power1234	power	power
5	kong-12	kong-	kong-
6	ark9999	ark	ark
7	love-1004	love-	love-

이하 생략

☞ 빈문자열을 전달하여 숫자를 모두 삭제 처리

## 예제) ID에서 특수기호 삭제

SQL1 \*

```

1 SELECT ID,
2      REGEXP_REPLACE(ID, '\w', '') AS RESULT1,
3      REGEXP_REPLACE(ID, '\w|_+', '') AS RESULT2,
4      REGEXP_REPLACE(ID, '[:punct:]', '') AS RESULT3
5 FROM PROFESSOR;

```

Result

ID	RESULT1	RESULT2	RESULT3
1 loveu	loveu	loveu	loveu
2 a1234	a1234	a1234	a1234
3 putty	putty	putty	putty
4 power1234	power1234	power1234	power1234
5 kong-12	kong12	kong12	kong12
6 ark9999	ark9999	ark9999	ark9999
7 love-1004	love1004	love1004	love1004

이하 생략

☞ \w 는 문자와 숫자, \_를 포함, \W 는 \w 의 반대 집합이므로 문자와 숫자와 \_가 아닌 특수기호와 공백을 의미

## 예제) PROFESSOR 테이블의 ID에서 문자와 문자 바로 뒤에 오는 숫자를 삭제(대소구분 X)

SQL1 \*

```

1 SELECT ID,
2      REGEXP_REPLACE(ID, '[a-z][0-9]') AS 성공1,
3      REGEXP_REPLACE(ID, '[a-zA-Z][0-9]') AS 성공2,
4      REGEXP_REPLACE(ID, '[A-Z][0-9]') AS 성공3,
5      REGEXP_REPLACE(ID, '[A-Z0-9]') AS 실패1,
6      REGEXP_REPLACE(ID, '[a-z][0-9]', '', 1, 0, 'i') AS 성공4
7 FROM PROFESSOR;

```

Result

ID	성공1	성공2	성공3	실패1	성공4
1 loveu	loveu	loveu	loveu		loveu
2 a1234	234	234	234		234
3 putty	putty	putty	putty		putty
4 power1234	powe234	powe234	powe234		powe234
5 kong-12	kong-12	kong-12	kong-12	-	kong-12
6 ark9999	ar999	ar999	ar999		ar999
7 love-1004	love-1004	love-1004	love-1004	-	love-1004

이하 생략

## \*\* kong-12 에서 g-1 을 지우는 방법

SQL1 \*

```

1 SELECT ID,
2      REGEXP_REPLACE(ID, '[A-Z]-[0-9]') AS 성공1,
3      REGEXP_REPLACE(ID, '[A-Z](-|_) [0-9]') AS 성공2
4 FROM PROFESSOR;

```

Result

ID	성공1	성공2
1 loveu	loveu	loveu
2 a1234	a1234	a1234
3 putty	putty	putty
4 power1234	power1234	power1234
5 kong-12	kon2	kon2
6 ark9999	ark9999	ark9999
7 love-1004	lov004	lov004

이하 생략

예제) PRODUCT 테이블의 상품명에서 괄호포함, 괄호안에 들어가는 모든 글자를 삭제

### \*\* 테이블 데이터

SQL1 \*

```

1 SELECT *
2 FROM PRODUCT;

```

Result

상품번호	상품명	조사일	판매가격	판매업소	판매지점
1	1 머거본 끌땅콩(135g)	2023/08/06 00:00:00	2980	GS	더프레시여의도점
2	2 무가염버터(450g)	2023/08/06 00:00:00	10300	GS	더프레시종로평창점
3	3 매일유업 뼈로가는 칼슘치즈(270g)	2023/08/06 00:00:00	7400	롯데슈퍼	G보라점
4	4 양반 들기름이 그윽한 김	2023/08/06 00:00:00	4690	롯데슈퍼	G복대점

### \*\* 정답

SQL1 \*

```

1 SELECT DISTINCT REGEXP_REPLACE(상품명, '\(.+\)', '')
2 FROM PRODUCT;

```

Result

REGEXP_REPLACE(상품명, '\(.+\)', '')
1 매일유업 뼈로가는 칼슘치즈
2 머거본 끌땅콩
3 무가염버터
4 양반 들기름이 그윽한 김

- ☞ 괄호는 서브그룹을 만드는 정규 표현식이므로 일반 괄호를 표현하기 위해서는 \) 로 전달해야 함
- ☞ \(.+\) : ()안에 엔터를 제외한 모든 값 허용

예제) REGEXP\_REPLACE 를 사용하여 두 번째 발견된 문자 값을 X로 치환

```
SQL1 *
1 ▶ SELECT NAME,
2      REGEXP_REPLACE(NAME, '[[:alpha:]]', '') AS RESULT1,
3      REGEXP_REPLACE(NAME, '[[:alpha:]]', '', 1, 1) AS RESULT2,
4      REGEXP_REPLACE(NAME, '[[:alpha:]]', 'x', 1, 2) AS RESULT3
5  FROM STUDENT;
```

Result

	NAME	RESULT1	RESULT2	RESULT3
1	강성근	성근	강X근	
2	고대영	대영	고X영	
3	권정현	정현	권X현	
4	김건일	건일	김X일	
5	김재현	재현	김X현	
6	나세희	세희	나X희	
7	박주현	주현	박X현	

이하 생략

- ☞ 문자 삭제 시 원하는 찾고자 하는 시작 위치와 발견횟수를 전달 할 수 있음
- ☞ RESULT3 : 처음부터 스캔하여 두 번째로 발견되는 문자를 X로 치환(마스킹 처리)

## ● REGEXP\_SUBSTR

- 정규식 표현식을 사용한 문자열 추출
- 옵션은 REGEXP\_REPLACE 와 동일

### \*\* 문법

REGEXP\_SUBSTR(대상, 패턴, [검색위치], [발견횟수], [옵션], [추출그룹])

### \*\* 특징

- 검색위치 생략 시 1
- 발견횟수 생략 시 1
- 추출그룹은 서브패턴을 추출 시 그 중 추출할 서브패턴 번호

예제) 전화번호를 분리하여 지역번호 추출

```
SQL1 *
1 ▶ SELECT TEL,
2      REGEXP_SUBSTR(TEL,
3      '(\d+)\)(\d+)-(\d+)',      --- 원본
4      1,                          --- 패턴(서브패턴과함께표현)
5      1,                          --- 시작위치
6      null,                      --- 발견횟수
7      1) AS 지역번호           --- 옵션
8  FROM STUDENT;                --- 추출할 서브패턴(그룹)번호
```

Result

TEL	지역번호
1 02)399-3947	02
2 031)304-3047	031
3 032)394-3048	032
4 02)3048-3039	02
5 051)304-3048	051
6 041)204-0383	041
7 02)4933-0382	02

이하 생략

- 전화번호 구성 : 숫자여러개 + ) + 숫자여러개 + - + 숫자여러개  
차례대로 \d+, \), \d+, -, \d+ 로 표현 가능, 그 중 첫 번째 그룹을 추출

예제) 이메일 아이디 추출(서브패턴 활용)

```
SQL1 *
1 SELECT EMAIL,
2      REGEXP_SUBSTR(EMAIL, '(.+@.+', 1, 1, NULL, 1) AS EMAIL_ID1,
3      REGEXP_SUBSTR(EMAIL, '(\w|\W)+@[a-z.]+', 1, 1, NULL, 1) AS EMAIL_ID2
4 FROM PROFESSOR;
```

Result

	EMAIL	EMAIL_ID1	EMAIL_ID2
1	loveu@abc.net	loveu	loveu
2	a1234@abc.net	a1234	a1234
3	pman@power.com	pman	pman
4	power1234@hamail.net	power1234	power1234
5	kong-12@naver.com	kong-12	kong-12
6	bdragon@naver.com	bdragon	bdragon
7	love-1004@hanmir.com	love-1004	love-1004

이하 생략

- EMAIL 주소는 EMAIL\_ID@ENGINE 으로 구성
- EMAIL\_ID : 몇 특수기호를 제외한 영문, 숫자, 기호로 구성
- ENGINE : 영문과 .으로 구성

### 3. REGEXP\_INSTR

- 주어진 문자열에서 특정패턴의 시작 위치를 반환

#### \*\* 문법

REGEXP\_INSTR(원본, 찾을패턴, [시작위치], [발견횟수], [출력옵션], [옵션], [추출그룹])

#### \*\* 출력옵션

- 1) 0 (default) : 문자열의 시작위치 리턴
- 2) 1 : 문자열의 끝위치 리턴

#### \*\* 특징

- 시작위치 생략 시 처음부터 확인(기본값:1)
- 발견횟수 생략 시 처음 발견된 문자열 위치 리턴

### 예제) ID 값에서 두 번째 발견된 숫자의 위치

```
SQL1 *
1 SELECT ID,
2      REGEXP_INSTR(ID, '\d', 1, 2)
3   FROM PROFESSOR;
```

Result

ID	REGEXP_INSTR(ID, '\d', 1, 2)
1 loveu	0
2 a1234	3
3 putty	0
4 power1234	7
5 kong-12	7
6 ark9999	5
7 love-1004	7

이하 생략

☞ \d는 숫자를 나타내는 표현이고, 뒤에 횟수를 지정하지 않으면 한 자리수의 숫자를 의미함

### 예제) 정규식 표현식을 사용한 패턴에 일치하는 n 번째 문자열 위치

```
SQL1 *
1 SELECT REGEXP_INSTR('500 ORACLE PARKWAY, REDWOOD SHORES, CA',
2                      '[^ ]+', 1, 2) AS "REGEXP_INSTR"
3   FROM DUAL;
```

Result

REGEXP_INSTR
1
5

☞ 다음과 같은 문자열에서 공백이 아닌 문자열의 반복들 중 처음부터 스캔하여 두 번째 발견된 것의 위치 리턴

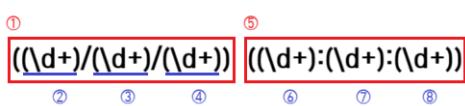
### 예제) 서브그룹에 대한 위치 출력(2024 기출)

```
SQL1 *
1 SELECT REGEXP_INSTR('2025/01/01 12:50:23',
2                      '(\d+)/(\d+)/(\d+)) ((\d+):(\d+):(\d+))',
3                      1,1,0,null,5) AS RESULT
4   FROM DUAL;
```

Result

RESULT
1
12

☞ 아래와 같은 순서로 서브그룹의 번호가 정해지기 때문에 5 번째 서브그룹은 12:50:23 전체를 나타낸다. 이때 출력옵션(5 번째 인수)이 0 이므로 12:50:23 문자열의 시작인 1의 위치를 리턴하게 된다. (만약 출력옵션이 1 인 경우 12:50:23 문자열의 끝인 3의 위치가 리턴됨)



## ● REGEXP\_LIKE

- 주어진 문자열에서 특정패턴을 갖는 경우 반환(WHERE 절 사용만 가능)
- 옵션 REGEXP\_REPLACE 와 동일

### \*\* 문법

REGEXP\_LIKE(원본, 찾을문자열, [옵션])

예제) ID 값이 숫자로 끝나는 교수 정보 출력

SQL1 *						
Result						
<a href="#">Grid Result</a> <a href="#">Server Output</a> <a href="#">Text Output</a> <a href="#">Explain Plan</a> <a href="#">Statistics</a>						
PROFNO	NAME	ID	POSITION	PAY	HIREDATE	
1	1002	경윤영	a1234	조교수	380	1997/02/15 00:00:00
2	2001	노경우	power1234	전임강사	250	2001/08/11 00:00:00
3	2002	이용준	kong-12	조교수	350	1995/11/30 00:00:00
4	2003	장요한	ark9999	정교수	490	1994/12/20 00:00:00
5	3001	전홍범	love-1004	정교수	530	1999/11/02 00:00:00
6	3002	정성용	one_10	조교수	330	2001/07/04 00:00:00
7	3003	백승윤	hong99	전임강사	290	2002/03/24 00:00:00

이하 생략

## ● REGEXP\_COUNT

- 주어진 문자열에서 특정패턴의 횟수를 반환
- 옵션 REGEXP\_REPLACE 와 동일
- 시작위치 생략 시 처음부터 스캔

### \*\* 문법

REGEXP\_COUNT (원본, 찾을문자열, 시작위치, [옵션])

예제) ID 값에서의 숫자의 수

SQL1 *			
Result			
<a href="#">Grid Result</a> <a href="#">Server Output</a> <a href="#">Text Output</a> <a href="#">Explain Plan</a> <a href="#">Statistics</a>			
ID	RESULT1	RESULT2	
1 captin	0	0	
2 sweety	0	0	
3 powerman	0	0	
4 lamb1	1	1	
5 number1	1	1	
6 bluedragon	0	0	
7 angel1004	4	1	
8 naone10	2	1	

이하 생략

☞ \d 는 한 자리수의 숫자를 의미하며 \d+는 연속적인 숫자를 의미. 따라서 COUNT 시 연속적인 숫자를 하나로 취급함

## 2-17. DML

### ● DML(Data Manipulation Language)

- 데이터의 삽입(INSERT), 수정(UPDATE), 삭제(DELETE), 병합(MERGE)
- 저장(commit) 혹은 취소(rollback) 반드시 필요

### ● INSERT

- 테이블에 행을 삽입할 때 사용
- 한 번에 한 행만 입력가능(SQL Server. 여러 행 동시 삽입 가능)
- 하나의 컬럼에는 한 값만 삽입 가능
- 컬럼별 데이터타입과 사이즈에 맞게 삽입
- INTO 절에 컬럼명을 명시하여 일부 컬럼만 입력 가능. 작성하지 않은 컬럼은 NULL 이 입력됨
  - ☞ NOT NULL 컬럼의 경우 오류 발생
- 전체 컬럼에 대한 데이터 입력시 테이블명 뒤의 컬럼명 생략 가능

\*\* 문법

```
SQL1 *
1  INSERT INTO 테이블명 VALUES(값1, 값2, ...);      -- 전체 컬럼 값을 입력
2  INSERT INTO 테이블명(컬럼1, 컬럼2, ...) VALUES(값1, 값2, ...);  -- 선택한 컬럼만 데이터 입력
```

예제) 테이블에 데이터 INSERT(한 행씩)

< 테이블 구조 >

Column	Nullable	Type	Comment
1 NO		NUMBER(10)	
2 NAME		VARCHAR2(10)	
3 PRICE		NUMBER(10)	

< 여러 행 INSERT >

```
SQL1 *
1  INSERT INTO MERGE_OLD VALUES(1, 'AMERICANO', 1000);
2  INSERT INTO MERGE_OLD VALUES(2, 'LATTE', 2000);
3  INSERT INTO MERGE_OLD VALUES(3, 'MILK', 3000);
4  COMMIT;
```

- ☞ 테이블의 각 컬럼별 데이터타입과 사이즈에 맞게 입력
- ☞ 문자 컬럼에 숫자값 입력 가능(권장 X)
- ☞ 숫자 컬럼에 '001'처럼 숫자처럼 생긴 문자값 입력 가능(권장 X)

## &lt; 결과 &gt;

SQL1 \*

```
1 SELECT *
2 FROM MERGE_OLD;
```

Result

	NO	NAME	PRICE
1	1	AMERICANO	1000
2	2	LATTE	2000
3	3	MILK	3000

## 예제) 서브쿼리를 사용한 여러 행 INSERT

SQL1 \*

```
1 INSERT INTO EMP3(EMPNO, ENAME, DEPTNO)
2 SELECT EMPNO, ENAME, DEPTNO
3 FROM EMP
4 WHERE DEPTNO = 20;
```

Result

5 rows Inserted.

## 예제) INSERT 시 컬럼 명시 생략으로 인한 오류 발생

## &lt; 테이블 구조 &gt;

SQL1 \*

```
1 DESC EMP3;
```

Result

Column	Nullable	Type	Comment
1 EMPNO		NUMBER(4)	
2 ENAME		VARCHAR2(10)	
3 JOB		VARCHAR2(9)	
4 MGR		NUMBER(4)	
5 HIREDATE		DATE	
6 SAL		NUMBER(7,2)	
7 COMM		NUMBER(7,2)	
8 DEPTNO		NUMBER(2)	

## &lt; 오류 발생 &gt;

SQL1 \*

```
1 INSERT INTO EMP3 VALUES(2, 'B');
```

Result

ORA-00947: 값의 수가 충분하지 않습니다

☞ 테이블명 뒤에 INSERT 할 컬럼명을 명시하지 않으면 모든 컬럼 INSERT 시도하는데 입력된 값은 두 개 밖에 없으므로 에러!

## &lt; 해결 &gt;

SQL1 \*

```
1 INSERT INTO EMP3(EMPNO, ENAME) VALUES(2, 'B');
```

Result

1 rows Inserted.

## ● UPDATE

- 데이터 수정할 때 사용
- 컬럼 단위 수행
- **다중컬럼 수정 가능**

### \*\* 문법

#### 1) 단일컬럼 수정

```
SQL1 *
1 UPDATE 테이블명
2 SET 수정할컬럼명 = 수정값
3 WHERE 조건;
```

☞ WHERE 절로 수정 대상을 선택 가능

예제) AMERICANO 의 PRICE 을 1500 으로 변경

```
SQL1 *
1 UPDATE MERGE_OLD
2 SET PRICE = 1500
3 WHERE NAME = 'AMERICANO';

Result
1 rows Updated.
```

#### 2) 다중컬럼 수정

##### - 방법 1

```
SQL1 *
1 UPDATE 테이블명
2 SET 수정컬럼명1 = 수정값1, 수정컬럼명2 = 수정값2, ...
3 WHERE 조건;
```

예제) 3 번의 NAME 을 HOT\_MILK 로 PRICE 을 2500 으로 변경

```
SQL1 *
1 UPDATE MERGE_OLD
2 SET NAME = 'HOT_MILK', PRICE = 2500
3 WHERE NO = 3;

Result
1 rows Updated.
```

##### - 방법 2

```
SQL1 *
1 UPDATE 테이블명
2 SET (수정컬럼명1, 수정컬럼명2, ...) = (SELECT 수정값1, 수정값2 ... )
3 WHERE 조건;
```

\* 서브쿼리의 결과가 수정할 각 행의 값마다 하나씩 전달돼야 함

예제) 서브쿼리를 사용한 여러 컬럼 동시 수정

< 수정 전 >

```
SQL1 *
1 SELECT ENAME, SAL, COMM
2   FROM EMP
3 WHERE ENAME = 'SMITH';
```

Result

ENAME	SAL	COMM
SMITH	800	

```
SQL1 *
1 UPDATE EMP
2   SET (SAL, COMM) = (SELECT MAX(SAL), MAX(COMM)
3                      FROM EMP)
4 WHERE ENAME = 'SMITH';
5
6 SELECT ENAME, SAL, COMM
7   FROM EMP
8 WHERE ENAME = 'SMITH';
```

Result

ENAME	SAL	COMM
SMITH	5000	1400

☞ 서브쿼리 결과가 각 컬럼마다 한 값으로 정의되지 않으면 수행 불가!

### ● DELETE

- 데이터를 삭제할 때 사용
- 행 단위 실행

\*\*문법

```
SQL1 *
1 DELETE [FROM] 테이블명
2 [WHERE 조건];
```

☞ WHERE 절로 삭제할 행 선택 가능

예제) NO 가 3인 행 삭제

```
SQL1 *
1 DELETE MERGE_OLD
2 WHERE NO = 3;
```

Result

1 rows Deleted.
-----------------

☞ 1개의 행이 삭제되었다는 것을 알 수 있음

#### 4) MERGE

- 데이터 병합
- 참조 테이블과 동일하게 맞추는 작업(참조테이블의 데이터 입력, 참조테이블의 값으로 수정 등)
- ☞ INSERT, UPDATE, DELETE 작업을 동시에 수행**

\*\* 문법

```
SQL1 *
1 MERGE INTO 테이블명
2 USING 참조테이블
3   ON (연결조건)
4   WHEN MATCHED THEN
5     UPDATE
6       SET 수정내용
7     DELETE (조건)
8   WHEN NOT MATCHED THEN
9     INSERT VALUES(값1, 값2, ...);
```

예제) OLD 테이블을 NEW 테이블과 동일하게 MERGE 문 작성

#### STEP1) MERGE 할 TABLE 확인

\*\* MERGE\_NEW 데이터 확인

```
SQL1 *
1 SELECT *
2 FROM MERGE_NEW;
```

Result

NO	NAME	PRICE
1	AMERICANO	1000
2	LATTE	2500
3	MOCHA	3000

\*\* MERGE\_OLD 테이블 확인

```
SQL1 *
1 SELECT *
2 FROM MERGE_OLD;
```

Result

NO	NAME	PRICE
1	AMERICANO	1500
2	LATTE	2000

#### STEP2) MERGE 문 작성

```
SQL1 *
1 MERGE INTO MERGE_OLD M1
2   USING MERGE_NEW M2
3     ON (M1.NO = M2.NO)
4   WHEN MATCHED THEN
5     UPDATE
6       SET M1.PRICE = M2.PRICE
7   WHEN NOT MATCHED THEN
8     INSERT VALUES(M2.NO, M2.NAME, M2.PRICE);
```

Result

3 rows upserted.
------------------

- ☞ 수정할 테이블명을 MERGE INTO 절에 명시, 참조테이블을 USING 절 명시
- ☞ 두 테이블의 데이터를 참조할 참조 조건을 ON 절 명시(괄호 필수)
- ☞ UPDATE 문에서는 테이블명 명시 X
- ☞ SET 절의 왼쪽이 수정테이블, 오른쪽이 참조 테이블 컬럼
- ☞ INSERT 문에는 INTO 절 없이 VALUES로 참조 컬럼명 전달

### STEP3) MERGE 결과 확인

NO	NAME	PRICE
1	AMERICANO	1000
2	LATTE	2500
3	MOCHA	3000

- ☞ 처음 테이블에는 없던 MOCHA 행이 삽입, 기존 LATTE 가격이 2000에서 2500으로 수정됨

## 2-18. TCL

### ● TCL(Transaction Control Language)

- 트랜잭션 제어어로 COMMIT, ROLLBACK 이 포함됨
- DML에 의해 조작된 결과를 작업단위(트랜잭션) 별로 제어하는 명령어
- DML 수행 후 트랜잭션을 정상 종료하지 않는 경우 LOCK 발생할 수 있음

#### \* 잠금(LOCK)

- 트랜잭션이 수행하는 동안 특정 데이터에 대해서 다른 트랜잭션이 동시에 접근하지 못하도록 제한
- 잠금이 걸린 데이터는 잠금을 실행한 트랜잭션만이 접근 및 해제 가능(관리자 권한 계정 제외)

### ● 트랜잭션

- 트랜잭션은 데이터베이스의 논리적 연산 단위(하나의 연속적인 업무 단위)
- 하나의 트랜잭션에는 하나 이상의 SQL 문장이 포함
- 분할 할 수 없는 최소의 단위
- ALL OR NOTHING 개념(모두 COMMIT하거나 ROLLBACK 처리해야 함)

#### \* 트랜잭션의 특성

- 원자성(atomicity) : 트랜잭션 정의된 연산들 모두 성공적으로 실행되던지 아니면 전혀 실행되지 않은 상태로 남아 있어야 함
- 일관성(consistency) : 트랜잭션 실행 전 데이터베이스 내용이 잘못되어 있지 않다면 트랜잭션 실행 이후에도 데이터베이스 내용의 잘못이 있으면 안됨

- 고립성(isolation) : 트랜잭션 실행 도중 다른 트랜잭션의 영향을 받아 잘못된 결과를 만들어서는 안됨
- 지속성(durability) : 트랜잭션이 성공적으로 수행되면 생성한 데이터베이스 내용이 영구적으로 저장

## ● COMMIT

- 입력, 수정, 삭제한 데이터에 이상이 없을 경우 데이터를 저장하는 명령어
- 한 번 COMMIT을 수행하면 **COMMIT 이전에 수행된 DML은 모두 저장되며 되돌릴 수 없음**
- ORACLE은 DDL 시 AUTO COMMIT(23c 버전부터 비활성화 가능) 있지만 SQL Server는 AUTO COMMIT 비활성화 설정 가능

## ● ROLLBACK

- 테이블 내 입력한 데이터나 수정한 데이터, 삭제한 데이터에 대해 변경을 취소하는 명령어
- 데이터베이스에 저장되지 않고 **최종 COMMIT 지점/변경 전/특정 SAVEPOINT 지점으로 원복됨**
- **최종 COMMIT 시점 이전까지 ROLLBACK 가능**
- SAVEPOINT를 설정하여 최종 COMMIT 시점이 아닌, 그 이후의 원하는 시점으로의 원복 가능

### \* SAVEPOINT

- 트랜잭션 내에서 룰백을 부분적으로 수행하기 위해 사용되는 지점을 지정하는 데 사용
- 사용자가 원하는 위치에 원하는 이름으로 설정 가능
- ROLLBACK TO SAVEPOINT\_NAME으로 원하는 지점으로 원복 가능(단, COMMIT 이전으로는 원복 불가)

### \*\* 문법

```
SQL1 *
1  SAVEPOINT SAVEPOINT_NAME;
```

### 예제) COMMIT과 ROLLBACK 후 최종 데이터 상태

<원본 테이블 데이터>

```
SQL1 *
1 > SELECT *
2   FROM ROLLBACK_TEST;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
-------	-------	-----	-----	----------	-----	------	--------

## &lt; 여러 DML 수행 &gt;

```

SQL1 *
1  INSERT INTO ROLLBACK_TEST(EMPNO, ENAME, DEPTNO) VALUES(9999, 'HONG', 10);
2  INSERT INTO ROLLBACK_TEST(EMPNO, ENAME, DEPTNO) VALUES(9998, 'PARK', 20);
3  COMMIT;
4
5  UPDATE ROLLBACK_TEST SET SAL = 3000 WHERE EMPNO = 9999;
6  ROLLBACK;
7
8  UPDATE ROLLBACK_TEST SET SAL = 3000 WHERE EMPNO = 9998;
9  COMMIT;
10
11 UPDATE ROLLBACK_TEST SET SAL = 4000 WHERE EMPNO = 9999;
12 SAVEPOINT A;
13
14 DELETE ROLLBACK_TEST WHERE EMPNO = 9999;
15 UPDATE ROLLBACK_TEST SET SAL = 9000 WHERE EMPNO = 9998;
16 ROLLBACK TO A;
17 COMMIT;

```

## &lt; 조회 결과 &gt;

```

SQL1 *
1 ▶ SELECT *
2   FROM ROLLBACK_TEST;
```

Result

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	9999	HONG			4000		10
2	9998	PARK			3000		20

☞ SAVEPOINT 이전 수행한 UPDATE 는 취소되지 않음!

## 2-19. DDL

## ● DDL(Data Definition Language)

- 데이터 정의어
- 데이터 구조 정의(객체 생성, 삭제, 변경) 언어
- CREATE(객체 생성), ALTER(객체 변경), DROP(객체 삭제), TRUNCATE(데이터 삭제)
- AUTO COMMIT(명령어 수행하면 즉시 저장, 원복 불가)

## ● CREATE

- 테이블이나 인덱스와 같은 객체를 생성하는 명령어
- 테이블 생성 시 테이블명, 컬럼명, 컬럼순서, 컬럼크기, 컬럼의 데이터타입 정의 필수
- 테이블 생성 시 각 컬럼의 제약조건 및 기본값은 생략 가능
- 테이블 생성 시 소유자 명시 가능(생략 시 명령어 수행 계정 소유)
- 숫자컬럼의 경우 컬럼 사이즈 생략 가능(날짜 컬럼은 사이즈 명시 X)

## \*\*문법 1

```
SQL1 *
1 CREATE TABLE [소유자.]테이블명(
2   컬럼1 데이터타입 [DEFAULT 기본값] [제약조건],
3   컬럼2 데이터타입 [DEFAULT 기본값] [제약조건],
4   ...
5 );
```

## \*\*문법 2(테이블 복제)

```
SQL1 *
1 CREATE 테이블명
2 AS
3 SELECT *
4   FROM 복제테이블명;
```

## \*\* 특징

- 복제테이블의 컬럼명과 컬럼의 데이터 타입이 복제됨
- SELECT 문에서 컬럼별칭 사용 시 컬럼별칭 이름으로 생성
- CREATE 문에서 컬럼명 변경 가능
- NULL 속성도 복제됨
- 테이블에 있는 제약조건, INDEX 등은 복제되지 X

## ● 데이터타입

데이터 타입	설명
CHAR(n)	고정형 문자 타입으로 사이즈 전달 필수, 사이즈 만큼 확정형 데이터가 입력됨(빈자리수는 공백으로 채워짐)
VARCHAR2(n)	가변형 문자타입으로 사이즈 전달 필수, 사이즈보다 작은 문자값이 입력되더라고 입력값 그대로 유지
NUMBER(p, s)	숫자형 타입으로 자리수 생략 가능, 소수점 자리 제한 시 s 전달(p는 총 자리수)
DATE	날짜타입으로 사이즈 전달 불가

☞ SQL Server 의 경우도 유사, VARCHAR2 -> VARCHAR 사용, NUMBER -> NUMERIC 사용

☞ SQL Server 의 경우 문자타입도 사이즈 생략 가능(생략 시 1)

## \* NUMBER(7,2)의 경우 총 자리수가 7을 초과할 수 없음

```
SQL1 *
1 INSERT INTO TEST1(B) VALUES(12345.67);      -- 입력 가능
2 INSERT INTO TEST1(B) VALUES(1234567.78);    -- 입력불가(사이즈초과)
```

## 예제) MERGE\_OLD 테이블 만들기

```
SQL1 *
1 CREATE TABLE MERGE_OLD(
2   NO NUMBER,
3   NAME VARCHAR2(10),
4   PRICE NUMBER
5 );
```

### 예제) EMP 테이블을 복제하여 TEST 테이블 만들기

```
SQL1 *
1 CREATE TABLE TEST
2 AS
3 SELECT *
4   FROM EMP;
```

### 예제) EMP 테이블 데이터 없이 구조만 복제

```
SQL1 *
1 CREATE TABLE TEST
2 AS
3 SELECT *
4   FROM EMP
5 WHERE 1=2;
```

- ☞ 항상 거짓인 조건을 SELECT 절에 전달하면, 데이터는 아무것도 출력되지 않지만 컬럼 정보들은 출력됨 따라서 테이블 내용은 제외하고 구조만 복제할 때 주로 사용!

### 예제) 테이블 복제 시 컬럼명 변경 가능

```
SQL1 *
1 CREATE TEST2(A,B)
2 AS
3 SELECT EMPNO, ENAME
4   FROM EMP;
```

### < 결과 >

Result			
Column	Nullable	Type	Comment
1 A		NUMBER(4)	
2 B		VARCHAR2(10)	

### ● ALTER

- 테이블 구조 변경(컬럼명, 컬럼 데이터타입, 컬럼 사이즈, DEFAULT 값, 컬럼 삭제, 컬럼 추가, 제약조건)
- 컬럼 순서 변경 불가(재생성으로 해결)

#### 1. 컬럼 추가

- 새로 추가된 컬럼위치는 맨 마지막(절대 중간 위치에 추가 불가)
- 컬럼 추가 시 데이터타입 필수, DEFAULT 값, 제약조건을 명시할 수 있음
- 여러 컬럼 동시 추가 가능(반드시 괄호 사용)

#### \*\* 문법)

```
SQL1 *
1 ALTER TABLE 테이블명 ADD 컬럼명 데이터타입 [DEFAULT] [제약조건];
```

### 예제) 컬럼 추가

SQL1 \*

```

1 CREATE TABLE EMP_T1
2 AS
3 SELECT *
4   FROM EMP;
5
6 ▶ SELECT *
7   FROM EMP_T1;

```

Result

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980/12/17 00:00:00	800		20
2	7499	ALLEN	SALESMAN	7698	1981/02/20 00:00:00	1600	300	30
3	7521	WARD	SALESMAN	7698	1981/02/22 00:00:00	1250	500	30
4	7566	JONES	MANAGER	7839	1981/04/02 00:00:00	2975		20
5	7654	MARTIN	SALESMAN	7698	1981/09/28 00:00:00	1250	1400	30
6	7698	BLAKE	MANAGER	7839	1981/05/01 00:00:00	2850		30
7	7782	CLARK	MANAGER	7839	1981/06/09 00:00:00	2450		10

이하 생략

\* 동시에 여러 컬럼을 추가할 경우 반드시 괄호와 함께 전달!

SQL1 \*

```

1 ALTER TABLE EMP_T1 ADD BIRTHDAY DATE;           -- 정상
2 ALTER TABLE EMP_T1 ADD (BIRTHDAY2 DATE);         -- 정상
3 ALTER TABLE EMP_T1 ADD (BIRTHDAY3 DATE, BIRTHDAY4 DATE); -- 정상
4 ▶ ALTER TABLE EMP_T1 ADD BIRTHDAY3 DATE, BIRTHDAY4 DATE; -- 여러(여러 컬럼 추가 시 반드시 괄호 필요)
5

```

Result

ORA-01735: 부적합한 ALTER TABLE 옵션입니다

\* 컬럼 추가시 NOT NULL 속성 전달 불가(컬럼 추가 시 모두 NULL 인 값을 갖고 추가되므로)

SQL1 \*

```

1 ▶ ALTER TABLE EMP_T1 ADD PHONE VARCHAR2(20) NOT NULL; -- 불가

```

Result

ORA-01758: 테이블은 필수 열을 추가하기 위해 (NOT NULL) 비어 있어야 합니다.

\* 컬럼 추가시 DEFAULT 을 선언하면 NOT NULL 속성을 갖는 컬럼 추가 가능

SQL1 \*

```

1 ▶ ALTER TABLE EMP_T1
2   ADD HPAGE VARCHAR2(30)
3   DEFAULT 'WWW.HDATALAB.CO.KR' NOT NULL;

```

\* 순서 주의(NOT NULL 은 DEFAULT 값 선언 뒤)

## 2. 컬럼(속성) 변경

- 컬럼 사이즈, 데이터타입, DEFAULT 값 변경 가능
- 여러 컬럼 동시 변경 가능

### \*\*문법

SQL1 \*

```

1 ALTER TABLE 테이블명 MODIFY(컬럼명 DEFAULT 값);

```

☞ 괄호 생략 가능

## 1) 컬럼 사이즈 변경

- 컬럼 사이즈 증가는 항상 가능
- 컬럼 사이즈 축소는 데이터 존재 여부에 따라 제한(데이터가 있는 경우 데이터의 최대 사이즈 만큼 축소 가능)
- [동시 변경 가능\(반드시 팔호 필요\)](#)

### 예제) 여러 컬럼 사이즈 수정

```
SQL1 *
1 ▶ ALTER TABLE TEST MODIFY (COL_A NUMBER(10), COL_B VARCHAR(6));
```

Result  
Statement Processed.

☞ 최대 길이보다 크거나 같은 사이즈로는 변경 가능

## 2) 데이터 타입 변경

- [비 컬럼일 경우 데이터 타입 변경 가능](#)
- [CHAR, VARCHAR 타입일 경우 데이터가 있어도 서로 변경 가능](#)

### 예제) 데이터타입 변경

#### < 테이블 구조 및 데이터 >

```
SQL1 *
1 ▶ DESC EMP_T2;
```

Result  
Grid Result Server Output Text Output Explain Plan Statistics

Column	Nullable	Type	Comment
1 ENAME		VARCHAR2(10)	
2 SAL		NUMBER(7,2)	
3 DEPTNO		VARCHAR2(10)	
4 HIREDATE		DATE	

```
SQL1 *
1 ▶ SELECT *
2   FROM EMP_T2;
```

Result  
Grid Result Server Output Text Output Explain Plan Statistics

NAME	SAL	DEPTNO	HIREDATE
KIM			2024/02/21 23:53:37
PARK			2024/02/21 23:56:31
CHOI	3000		2024/02/21 23:56:38

#### < 컬럼 데이터 타입 변경 >

```
SQL1 *
1 ▶ ALTER TABLE EMP_T2 MODIFY DEPTNO NUMBER(4); -- 가능
2 ▶ ALTER TABLE EMP_T2 MODIFY SAL VARCHAR2(4); -- 에러
```

Result  
ORA-01439: 데이터 유형을 변경할 열은 비어 있어야 합니다

### 예제) CHAR <-> VARCHAR 데이터타입 변경

#### < 테이블 구조 및 데이터 >

SQL1 \*

```
1 ▶ DESC STUDENT_T1;
```

Result

	Column	Nullable	Type	Comment
1	STUDNO		NUMBER(4)	
2	NAME	NOT NULL	VARCHAR2(20)	
3	SID	NOT NULL	VARCHAR2(30)	
4	GRADE		NUMBER	
5	JUMIN	NOT NULL	CHAR(13)	

SQL1 \*

```
1 ▶ SELECT *
2     FROM STUDENT_T1;
```

Result

	STUDNO	NAME	SID	GRADE	JUMIN	BIRTHDAY
1	5001	강성근	75true	4	9810071305173	1998/10/07
2	5002	고대영	pooh94	4	9809121384567	1998/09/12

#### < 컬럼 데이터 타입 변경 >

```
SQL1 *
1 ALTER TABLE STUDENT_T1 MODIFY JUMIN VARCHAR2(13); -- 가능
2 ▶ ALTER TABLE STUDENT_T1 MODIFY JUMIN CHAR(13); -- 가능
```

### 3) DEFAULT 값 변경

- DEFAULT 값이란 특정 컬럼에 값이 생략될 경우(입력 시 언급되지 않을 경우) 자동으로 부여되는 값
- INSERT 시 DEFAULT 값이 선언된 컬럼에 NULL을 직접 입력할 때는 DEFAULT 값이 아닌 NULL이 입력됨
- 이미 데이터가 존재하는 테이블에 DEFAULT 값 선언 시 기존 데이터 수정 안됨(이후 입력된 데이터부터 적용)
- DEFAULT 값 해제 시 DEFAULT 값을 NULL로 선언

### 예제) DEFAULT 값 변경 및 적용

#### < TEST DATA 생성 >

SQL1 \*

```
1 ▶ SELECT *
2     FROM EMP_T2;
```

Result

	NAME	SAL	DEPTNO	HIREDATE
1	KIM			2024/02/21 23:53:37

## &lt; DEFAULT 값 수정 &gt;

```
SQL1 *
1 ▶ ALTER TABLE EMP_T2 MODIFY (SAL DEFAULT 3000);
```

Result  
Statement Processed.

## &lt; 새로운 값 입력 &gt;

```
SQL1 *
1 INSERT INTO EMP_T2 VALUES('PARK', NULL, NULL, SYSDATE);
2 INSERT INTO EMP_T2(ENAME, DEPTNO, HIREDATE) VALUES('CHOI', NULL, SYSDATE);
3 COMMIT;
```

## &lt; 데이터 확인 &gt;

```
SQL1 *
1 ▶ SELECT *
2      FROM EMP_T2;
```

Result  
Grid Result Server Output Text Output Explain Plan Statistics

	NAME	SAL	DEPTNO	HIREDATE
1	KIM			2024/02/21 23:53:37
2	PARK			2024/02/21 23:56:31
3	CHOI	3000		2024/02/21 23:56:38

- ☞ PARK 는 SAL 값이 DEFAULT 값이 아닌 NULL로 입력됨
- ☞ CHOI 는 SAL 값이 DEFAULT 값으로 입력됨(입력 시 SAL 컬럼 언급 안됐기 때문)

## 3. 컬럼 이름 변경

- 항상 가능
- **동시 여러 컬럼 이름 변경 불가(괄호 전달 불가)**
- ALTER ... RENAME 명령어로 처리

## \*\* 문법

```
SQL1 *
1 ALTER TABLE TABLE_NAME RENAME COLUMN 기존컬럼명 TO 새컬럼명;
```

## 예제) 컬럼 이름 변경

## &lt; 테이블 구조 &gt;

```
SQL1 *
1 ▶ DESC EMP_T1;
```

Result  
Grid Result Server Output Text Output Explain Plan Statistics

	Column	Nullable	Type	Comment
1	EMPNO		NUMBER(4)	
2	ENAME		VARCHAR2(10)	
3	JOB		VARCHAR2(9)	
4	MGR		NUMBER(4)	

## &lt; 컬럼 이름 변경 &gt;

```
SQL1 *
1 ALTER TABLE EMP_T1 RENAME COLUMN ENAME TO NAME;
2 DESC EMP_T1;
```

Result

Column	Nullable	Type	Comment
1 EMPNO		NUMBER(4)	
2 NAME		VARCHAR2(10)	
3 JOB		VARCHAR2(9)	
4 MGR		NUMBER(4)	

## 4. 컬럼 삭제

- 데이터 존재 여부와 상관없이 언제나 가능
- RECYCLEBIN에 남지 X(FLASHBACK으로 복구 불가)
- 동시 삭제 불가

## 예제) COL\_A 컬럼의 삭제

```
SQL1 *
1 ALTER TABLE TEST DROP COLUMN COL_A;
```

Result

Statement Processed.

☞ 정상

## 예제) 2개이상 컬럼 삭제 시도 시 에러 발생

```
SQL1 *
1 ALTER TABLE TEST DROP COLUMN COL_A, COL_B;
```

Result

ORA-00933: SQL 명령어가 올바르게 종료되지 않았습니다

☞ 괄호로 묶어서 전달하여도 동시 삭제 불가

## ● DROP

- 객체(테이블, 인덱스 등) 삭제
- DROP 후에는 조회 불가

## \*\*문법

```
SQL1 *
1 DROP TABLE 테이블명 [PURGE];
```

\* PURGE로 테이블 삭제 시 RECYCLEBIN에서 조회 불가

### 예제) TEST 테이블 DROP 후 조회 결과

```
SQL1 *
1 ▶ DROP TABLE TEST;
```

Result  
Statement Processed.

```
SQL1 *
1 ▶ SELECT *
2   FROM TEST;
```

Result  
Grid Result Server Output Text Output Explain Plan Statistics

ORA-00942: 테이블 또는 뷰가 존재하지 않습니다

☞ 테이블 또는 뷰가 존재하지 않는다는 에러 발생

### ● TRUNCATE

- 구조 남기고 데이터만 즉시 삭제, 즉시 반영(AUTO COMMIT)
- RECYCLEBIN에 남지 않음

#### \*\*문법

```
SQL1 *
1 TRUNCATE TABLE 테이블명;
```

### 예제) TRUNCATE 사용하여 데이터 전부 삭제

```
SQL1 *
1 TRUNCATE TABLE TEST;
```

#### < 삭제 후 데이터 조회 >

```
SQL1 *
1 ▶ SELECT *
2   FROM TEST;
```

Result  
Grid Result Server Output Text Output Explain Plan Statistics

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
-------	-------	-----	-----	----------	-----	------	--------

☞ 구조(테이블 명, 컬럼 등)만 남고 데이터는 삭제된 것을 확인할 수 있음

### ● DELETE / DROP / TRUNCATE 차이

- DELETE : 데이터 일부 또는 전체 삭제, 롤백 가능
- TRUNCATE : 데이터 전체 삭제만 가능(일부 삭제 불가), 즉시 반영(롤백 불가)
- DROP : 데이터와 구조를 동시 삭제, 즉시 반영(롤백 불가)

## ● 제약조건

- 데이터 무결성을 위해 각 컬럼에 생성하는 데이터의 제약 장치
- 테이블 생성 시 정의 가능, 컬럼 추가 시 정의 가능, 이미 생성된 컬럼에 제약조건만 추가 가능

### 1. PRIMARY KEY(기본키)

- 유일한 식별자(각 행을 구별할 수 있는 식별자 기능)
- 중복 허용 X, NULL 허용 X => UNIQUE + NOT NULL
- 특정 컬럼에 PRIMARY KEY 생성하면 NOT NULL 속성 자동 부여(CTAS로 테이블 복사 시 복사되지 X)
- 하나의 테이블에 여러 기본키를 생성할 수 없음
- 하나의 기본키를 여러 컬럼을 결합하여 생성할 수 있음
- PRIMARY KEY 생성 시 자동으로 UNIQUE INDEX 생성

#### \*\*문법

##### 1) 테이블 생성 시 제약조건 생성

```
SQL1 *
1 CREATE TABLE 테이블명(
2   컬럼1 데이터타입 [DEFAULT 기본값] [[CONSTRAINT 제약조건명] 제약조건종류],
3   컬럼2 데이터타입 [DEFAULT 기본값] [[CONSTRAINT 제약조건명] 제약조건종류],
4   . . .
5   [[CONSTRAINT 제약조건명] 제약조건종류]
6 );
```

##### 2) 컬럼 추가 시 제약 조건 생성

```
SQL1 *
1 ALTER TABLE 테이블명
2 ADD 컬럼명 데이터타입 [DEFAULT 기본값] [[CONSTRAINT 제약조건명] 제약조건종류];
```

##### 3) 이미 생성된 컬럼에 제약조건만 추가

```
SQL1 *
1 ALTER TABLE 테이블명 ADD [CONSTRAINT 제약조건명] 제약조건종류;
```

##### 4) 제약조건 삭제

```
SQL1 *
1 ALTER TABLE 테이블명 DROP CONSTRAINT 제약조건명;
```

#### 예제) 테이블 생성 시 제약조건 설정(이름 전달 없이)

```
SQL1 *
1 CREATE TABLE TEST_1(NO NUMBER(10) PRIMARY KEY,
2                      NAME VARCHAR2(20));
```

☞ 제약조건 생성 시 이름을 설정하지 않으면 자동으로 부여

```
SQL1 *
1 CREATE TABLE TEST_2(
2 NO1 NUMBER,
3 NO2 NUMBER,
4 NAME VARCHAR2(10),
5 CONSTRAINT TEST2_PK PRIMARY KEY(NO1, NO2)
6 );
```

☞ CREATE 문 밑에 제약조건 이름과 함께 전달 가능!

예제) 테이블 생성 시 제약조건 설정(이름과 함께 전달)

```
SQL1 *
1 CREATE TABLE TEST_2 (NO NUMBER(10) CONSTRAINT TEST_NO_PK PRIMARY KEY,
2 NAME VARCHAR2(20));
```

예제) 컬럼 추가 시 제약조건 생성

```
SQL1 *
1 CREATE TABLE TEST_3 (SUBJECT VARCHAR2(20));
2 ALTER TABLE TEST_3 ADD NO NUMBER(10) PRIMARY KEY;
```

예제) 이미 있는 컬럼에 제약조건만 생성

```
SQL1 *
1 CREATE TABLE TEST_4 (NO NUMBER(10),
2 NAME VARCHAR2(20));
3 ALTER TABLE TEST_4 ADD CONSTRAINT TEST4_NO_PK PRIMARY KEY(NO);
```

## 2. UNIQUE

- 중복을 허용하지 않음
- NULL은 허용
- UNIQUE INDEX 자동 생성

예제) UNIQUE KEY 가 생성된 컬럼의 값 입력

< UNIQUE KEY 가 생성 >

```
SQL1 *
1 CREATE TABLE TEST_5(
2 NO NUMBER,
3 SUBJECT VARCHAR2(20) UNIQUE);
```

< 값의 입력 >

```
SQL1 *
1 INSERT INTO TEST_5 VALUES(1, 'A');      -- 정상
2 INSERT INTO TEST_5 VALUES(2, 'A');      -- 에러
3 INSERT INTO TEST_5 VALUES(2, NULL);     -- 정상
4 INSERT INTO TEST_5 VALUES(3, NULL);     -- 정상
```

## &lt; 확인 &gt;

SQL1 \*

```

1 SELECT *
2   FROM TEST_5;

```

Result

NO	SUBJECT
1	A
2	
3	

## 3. NOT NULL

- 다른 제약조건과 다르게 컬럼의 특징을 나타냄 => CTAS로 복제 시 따라감
- 컬럼 생성 시 NOT NULL을 선언하지 않으면 Nullable 컬럼으로 생성됨
- 이미 만들어진 컬럼에 NOT NULL 선언 시 제약조건 생성이 아닌 컬럼 수정(MODIFY)으로 해결

## 예제) NOT NULL 선언

SQL1 \*

```

1 ALTER TABLE PROFESSOR_TEST1 ADD NOT NULL(COL12);
2 ALTER TABLE PROFESSOR_TEST1 MODIFY COL12 NOT NULL;
3 ALTER TABLE PROFESSOR_TEST1 MODIFY COL12
4           CONSTRAINT PROFESSOR_COL12_NN NOT NULL; -- 제약조건 이름 부여

```

-- 불가  
-- 컬럼 수정으로 NOT NULL 처리

## 4. FOREIGN KEY

- 참조테이블의 참조 컬럼에 있는 데이터를 확인하면서 본 테이블 데이터를 관리할 목적으로 생성
- 반드시 참조(부모)테이블의 참조 컬럼(REFERENCE KEY)이 사전에 PK 혹은 UNIQUE KEY를 가져야 함!

## \*\*문법

SQL1 \*

```

1 CREATE TABLE 테이블명(
2   컬럼1 데이터타입 [DEFAULT 값] REFERENCES 참조테이블(참조키),
3   ...
4 );

```

## 예제) FOREIGN KEY 테스트

## \*\* 테스트 테이블 생성

SQL1 \*

```

1 CREATE TABLE EMP_TEST1 AS SELECT * FROM EMP;
2 CREATE TABLE DEPT_TEST1 AS SELECT * FROM DEPT;

```

## \*\* 부모 테이블(DEPT\_TEST1)에 REFERENCE KEY(참조대상)에 PK 설정!

SQL1 \*

```

1 ALTER TABLE DEPT_TEST1 ADD CONSTRAINT DEPT_TEST1_DEPTNO_PK PRIMARY KEY(DEPTNO);

```

\*\* 자식 테이블(EMP\_TEST1)에 FOREIGN KEY 생성

```
SQL1 *
1 ▶ ALTER TABLE EMP_TEST1 ADD CONSTRAINT EMP_TEST1_DEPTNO_FK
2      FOREIGN KEY(DEPTNO) REFERENCES DEPT_TEST1(DEPTNO);
```

생략 가능

TEST1) 자식 테이블(EMP\_TEST1)에서 10 번 부서원 삭제 시도

```
SQL1 *
1 ▶ DELETE EMP_TEST1 WHERE DEPTNO = 10; -- 가능
```

TEST2) 자식 테이블(EMP\_TEST1)에서 20 번 부서원 50 번으로 변경 시도(불가)

```
SQL1 *
1 ▶ UPDATE EMP_TEST1 SET DEPTNO = 50 WHERE DEPTNO = 20; -- 에러
```

Result

ORA-02291: 무결성 제약조건(SCOTT.EMP\_TEST1\_DEPTNO\_FK)이 위배되었습니다- 부모 키가 없습니다

**☞ 부모 테이블에 50 번 부서번호가 정의되어 있지 않아 자식 테이블에 해당 값으로 수정 불가!**

TEST3) 자식 테이블(EMP\_TEST1)에서 50 번 부서원(나머지 정보 자유) 입력 시도(불가)

```
SQL1 *
1 ▶ INSERT INTO EMP_TEST1(EMPNO, ENAME, DEPTNO) VALUES(1111, 'A', 50); -- 에러
```

Result

ORA-02291: 무결성 제약조건(SCOTT.EMP\_TEST1\_DEPTNO\_FK)이 위배되었습니다- 부모 키가 없습니다

Col : 1

**☞ 부모 테이블에 50 번 부서번호가 정의되어 있지 않아 자식 테이블에 해당 값으로 입력 불가!**

TEST4) 부모 테이블(DEPT\_TEST1)에 20 번 부서원 삭제 시도(불가)

```
SQL1 *
1 ▶ DELETE DEPT_TEST1 WHERE DEPTNO = 20; -- 에러
```

Result

ORA-02292: 무결성 제약조건(SCOTT.EMP\_TEST1\_DEPTNO\_FK)이 위배되었습니다- 자식 레코드가 발견되었습니다

**☞ 20 번 부서 정보가 자식 테이블에 존재하므로 삭제 불가**

TEST5) 부모 테이블(DEPT\_TEST1)에 20 번 부서원의 부서번호를 60 변경 시도(불가)

```
SQL1 *
1 ▶ UPDATE DEPT_TEST1 SET DEPTNO = 60 WHERE DEPTNO = 20; -- 에러
```

Result

ORA-02292: 무결성 제약조건(SCOTT.EMP\_TEST1\_DEPTNO\_FK)이 위배되었습니다- 자식 레코드가 발견되었습니다

**☞ 20 번 부서 정보가 자식 테이블에 존재하므로 다른 값으로 변경 불가**

● FOREIGN KEY 옵션(생성 시 정의, 변경 불가 -> 재생성)

1. **ON DELETE CASCADE** : 부모 데이터 삭제 시 자식 데이터 함께 삭제
2. **ON DELETE SET NULL** : 부모 데이터 삭제 시 자식 데이터의 참조값은 NULL로 수정

예제) FOREIGN KEY 옵션 TEST(ON DELETE CASCADE )

< FOREIGN KEY 재생성(ON DELETE CASCADE ) >

```
SQL1 *
1 ALTER TABLE EMP_TEST1 DROP CONSTRAINT EMP_TEST1_DEPTNO_FK;
2 ALTER TABLE EMP_TEST1 ADD CONSTRAINT EMP_TEST1_DEPTNO_FK
3 ▶      FOREIGN KEY(DEPTNO) REFERENCES DEPT_TEST1(DEPTNO) ON DELETE CASCADE;
```

Result

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1 7782	CLARK	MANAGER	7839	1981/06/09 00:00:00	2450		10
2 7839	KING	PRESIDENT		1981/11/17 00:00:00	5000		10
3 7934	MILLER	CLERK	7782	1982/01/23 00:00:00	1300		10

< 부모 데이터 삭제 >

```
SQL1 *
1 DELETE DEPT_TEST1 WHERE DEPTNO = 10;
2 ▶ SELECT * FROM EMP_TEST1 WHERE DEPTNO = 10;
```

Result

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
-------	-------	-----	-----	----------	-----	------	--------

☞ 부모 데이터 삭제 시, 자식 데이터도 함께 삭제됨!

예제) FOREIGN KEY 옵션 TEST(ON DELETE SET NULL)

< FOREIGN KEY 재생성(ON DELETE SET NULL ) >

```
SQL1 *
1 ALTER TABLE EMP_TEST1 DROP CONSTRAINT EMP_TEST3_DEPTNO_FK;
2 ALTER TABLE EMP_TEST1 ADD CONSTRAINT EMP_TEST3_DEPTNO_FK
3      FOREIGN KEY(DEPTNO) REFERENCES DEPT_TEST1(DEPTNO) ON DELETE SET NULL;
4 ▶ SELECT * FROM EMP_TEST1 WHERE JOB = 'MANAGER';
```

Result

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1 7566	JONES	MANAGER	7839	1981/04/02 00:00:00	2975		20
2 7698	BLAKE	MANAGER	7839	1981/05/01 00:00:00	2850		30

## &lt; 부모 데이터 삭제 &gt;

```
SQL1 *
1 DELETE DEPT_TEST1 WHERE DEPTNO = 30;
2 SELECT * FROM EMP_TEST1 WHERE JOB = 'MANAGER';
```

Result

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7566	JONES	MANAGER	7839	1981/04/02 00:00:00	2975		20
2	7698	BLAKE	MANAGER	7839	1981/05/01 00:00:00	2850		

☞ 자식 테이블의 데이터도 함께 삭제되지 않음(NULL로 수정)

## 5. CHECK

- 직접적으로 데이터의 값 제한
- ex) 양수(1, 2, 3, 4) 중 하나

예제) EMP\_TEST1 테이블의 SAL 값은 0 이상이어야 한다는 CHECK 제약조건 추가

```
SQL1 *
1 ALTER TABLE EMP_TEST1 ADD CONSTRAINT EMP_TEST1_SAL_CK CHECK (SAL > 0);
```

Result

## ● 기타 오브젝트

## 1) 뷰(VIEW)

- 저장공간을 가지지는 않지만 테이블처럼 조회 및 수정할 수 있는 객체

## \*\* 뷰(VIEW)의 종류

- 단순뷰 : 하나의 테이블 조회 뷰(VIEW)
- 복합뷰 : 두 이상의 테이블 조인 뷰(VIEW)

## \*\* 뷰(VIEW)의 특징

- 뷰(VIEW)는 기본 테이블로부터 유도된 테이블이기에 기본 테이블과 같은 형태의 구조를 가지고 있으며, 조작도 기본 테이블과 거의 같음
- 뷰는 가상의 테이블이기에 물리적으로 구현되어 있지 않으며 저장공간을 차지하지 않음
- 데이터를 안전하게 보호 가능
- 이미 정의되어 있는 뷰(VIEW)는 다른 뷰(VIEW)의 정의에 기초가 될 수 있음
- 기본 테이블이 삭제되면 그 테이블을 참조하여 만든 뷰 역시 삭제됨

## \*\* 뷰(VIEW)의 장점

- 논리적 독립성을 제공
- 데이터의 접근을 제어 함으로써 보안유지
- 사용자의 데이터 관리 단순화
- 데이터의 다양한 지원 가능

## \*\* 뷰(VIEW)의 단점

- 뷰의 정의 변경 불가
- 삽입, 삭제, 갱신 연산에 제한
- 인덱스 구성불가

## \*\* 문법

```
SQL1 *
1 CREATE [OR REPLACE] VIEW 뷰이름
2 AS
3 조회쿼리;
```

## \*\* 뷰(VIEW)의 삭제

```
SQL1 *
1 DROP VIEW 뷰명;
```

## 예제) 뷰 생성 및 조회

### < 생성 >

```
SQL1 *
1 ▶ CREATE VIEW VIEW_EMP_DEPT
2   AS
3     SELECT E.EMPNO, E.ENAME, E.SAL, E.DEPTNO, D.DNAME
4       FROM EMP E, DEPT D
5      WHERE E.DEPTNO = D.DEPTNO;
```

Result  
Statement Processed.

☞ EMP 테이블과 DEPT 테이블에서 가져올 데이터를 선택

### < 조회 >

```
SQL1 *
1 ▶ SELECT *
2   FROM VIEW_EMP_DEPT;
```

Result

	EMPNO	ENAME	SAL	DEPTNO	DNAME
1	7782	CLARK	2450	10	ACCOUNTING
2	7839	KING	5000	10	ACCOUNTING
3	7934	MILLER	1300	10	ACCOUNTING
4	7566	JONES	2975	20	RESEARCH
5	7902	FORD	3000	20	RESEARCH
6	7876	ADAMS	1100	20	RESEARCH
7	7369	SMITH	5000	20	RESEARCH

## 2) 시퀀스(SEQUENCE)

- 자동으로 연속적인 숫자 부여해주는 객체

\*\* 문법

```
SQL1 *
1 CREATE SEQUENCE 시퀀스명
2 INCREMENT BY
3 START WITH
4 MAXVALUE
5 MINVALUE
6 CYCLE | NOCYCLE
7 CACHE N
8 ;
```

-- 증가값(DEFAULT : 1)  
-- 시작값(DEFAULT : 1)  
-- 마지막값(증가시퀀스), 재사용시 시작값(감소시퀀스)  
-- 재사용시 시작값(증가시퀀스), 마지막값(감소시퀀스)  
-- 시퀀스 번호 재사용(DEFAULT : NOCYCLE)  
-- 캐시값(DEFAULT : 20)

## 3) 시노님(SYNONYM)

- 테이블 별칭 생성  
 EX) HR 계정에서 SCOTT.EMP 를 EMP 로 조회하는 방법

\*\* 문법

```
SQL1 *
1 CREATE [OR REPLACE] [PUBLIC] SYNONYM 별칭 FOR 테이블명;
```

- ☞ OR REPLACE : 기존에 같은 이름으로 시노님이 생성되어 있는 경우 대체
- ☞ PUBLIC : 시노님을 생성한 유저만 사용 가능한 PRIVATE SYNONYM 의 반대(누구나 사용가능)
- ☞ PUBLIC 으로 생성한 시노님은 반드시 PUBLIC 으로 삭제

예제) 시노님 생성 전 후 테이블 조회비교(HR 계정으로 조회)

< 시노님 생성 >

```
SQL1 *
1 ▶ CREATE PUBLIC SYNONYM EMP FOR SCOTT.EMP;
```

< 시노님 생성 전 HR 계정에서 EMP 조회 >

```
SQL1 *
1 ▶ SELECT *
2      FROM EMP;
```

Result

Grid Result Server Output Text Output Explain Plan Statistics

Error: # 942, ORA-00942: 테이블 또는 뷰가 존재하지 않습니다

## &lt; 시노님 생성 후 HR 계정에서 EMP 조회 &gt;

SQL1 \*

```
1 SELECT *
2 FROM EMP;
```

Result

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980/12/17 00:00:00	800		20
2	7499	ALLEN	SALESM	7698	1981/02/20 00:00:00	1...	300	30

## 2-20. DCL

## ● DCL(Data Control Language)

- 데이터 제어어로 객체에 대한 권한을 부여(GRANT)하거나 회수(REVOKE)하는 기능
- 테이블 소유자는 타계정에 테이블 조회 및 수정 권한 부여 및 회수 가능

## ● 권한

- 일반적으로 본인(접속한 계정) 소유가 아닌 테이블은 원칙적으로 조회 불가(권한 통제)
- 업무적으로 필요시 테이블 소유자가 아닌 계정에 테이블 조회, 수정 권한 부여 가능

## ※ 권한 종류

## 1) 오브젝트권한

- 테이블에 대한 권한 제어
- ex) 특정 테이블에 대한 SELECT, INSERT, UPDATE, DELETE, MERGE 권한
- 테이블 소유자는 타계정에 소유 테이블에 대한 조회 및 수정 권한 부여 및 회수 가능

## 2) 시스템권한

- 시스템 작업(테이블 생성 등) 등을 제어
- ex) 테이블 생성 권한, 인덱스 삭제 권한
- 관리자 권한만 권한 부여 및 회수 가능

## ● GRANT

- 권한 부여 시 반드시 테이블 소유자나 관리자계정(SYS, SYSTEM)으로 접속하여 권한을 부여하여야 함
- 동시에 여러 유저에 대한 권한 부여 가능
- 동시 여러 권한 부여 가능
- 동시 여러 객체 권한 부여 불가

## \*\* 문법

SQL1 \*

```
1 GRANT 권한 ON 테이블명 TO 유저;
```

### 예제) 오브젝트 권한 부여(PROFESSOR 소유자 실행)

SQL1 \*

```

1 GRANT SELECT ON PROFESSOR TO HR;          -- 가능
2 GRANT SELECT ON PROFESSOR TO ORDDATA, BI;  -- 가능
3 GRANT SELECT, UPDATE, INSERT ON PROFESSOR TO HR;  -- 가능
4 ▶ GRANT SELECT ON PROFESSOR, DEPT2 TO HR;    -- 예외

```

Result

Grid Result Server Output Text Output Explain Plan Statistics

ORA-00905: 누락된 키워드 Ln 4, Col 61 0.00 sec.

### 예제) 시스템 권한 부여(관리자 권한으로 실행)

SQL1 \*

```

1 GRANT CREATE TABLE TO HR;          -- 가능
2 GRANT CREATE TABLE TO HR, BI;    -- 가능
3 ▶ GRANT CREATE TABLE, DROP ANY TABLE TO HR;  -- 가능

```

Result

Statement Processed.

## ● REVOKE

- 동시 여러 권한 회수 가능
- 이미 회수된 권한 재회수 불가
- 동시 여러 유저로부터의 권한 회수 가능

### \*\* 문법

SQL1 \*

```

1 REVOKE 권한 ON 테이블명 FROM 유저;

```

### 예제) 오브젝트 권한 회수

SQL1 \*

```

1 REVOKE SELECT, ON UPDATE, INSERT ON PROFESSOR FROM HR;  -- 가능
2 REVOKE SELECT ON PROFESSOR FROM ORDDATA, BI;           -- 가능
3 REVOKE SELECT ON PROFESSOR FROM HR, BI;                  -- 예외(이미 회수된 권한 재회수 불가)

```

Result

ORA-01927: 허가하지 않은 권한을 REVOKE할 수 없습니다 Col : 1 Ln 3, Col 86 0.00 sec.

## ● 룰(ROLE)

- 권한의 묶음(생성 가능한 객체)
- SYSTEM 계정에서 ROLE 생성 가능

### \*\* 문법

SQL1 \*

```

1 CREATE ROLE 룰이름;

```

### 예제) 룰(ROLE)

< 생성 >

```
SQL1 *
1 ▶ CREATE ROLE ROLE_SEL;

Result
Statement Processed.
```

< 룰에 권한 담기 >

```
SQL1 *
1 GRANT SELECT ON EMP TO ROLE_SEL;
2 GRANT SELECT ON STUDENT TO ROLE_SEL;
3 GRANT SELECT ON DEPT TO ROLE_SEL;
4 ▶ GRANT SELECT ON DEPARTMENT TO ROLE_SEL;
```

< 룰 부여 >

```
SQL1 *
1 ▶ GRANT ROLE_SEL TO HR;

Result
Statement Processed.
```

< HR 계정에서 수행 >

```
SQL1 *
1 ▶ SELECT *
2   FROM SCOTT.DEPT;

Result
Grid Result Server Output Text Output Explain Plan Statistics
DEPTNO DNAME LOC
1      10 ACCOUNTING NEW YORK
2      20 RESEARCH DALLAS
3      30 SALES CHICAGO
4      40 OPERATIONS BOSTON
```

< 룰에서 권한 빼기 >

```
SQL1 *
1 ▶ REVOKE SELECT ON DEPARTMENT FROM ROLE_SEL;
```

< 권한 회수 후 HR 계정에서 조회 >

```
SQL1 *
1 ▶ SELECT * FROM SCOTT.DEPARTMENT;

Result
ORA-01031: 권한이 불충분합니다
```

☞ 권한이 불충분하다는 에러

☞ ROLE에서 회수된 권한은 즉시 반영되므로 다시 ROLE을 부여할 필요가 없음

< 롤을 통해 부여한 권한 직접 회수(SCOTT에서 실행) >

```
SQL1 *
1 ► REVOKE SELECT ON STUDENT FROM HR;

Result
ORA-01927: 허가하지 않은 권한을 REVOKE할 수 없습니다
```

- ☞ ROLE을 통해 부여한 권한은 직접 회수 불가
- ☞ ROLE을 통한 회수만 가능

## ● 권한부여 옵션(중간관리자의 권한)

### 1. WITH GRANT OPTION

- WITH GRANT OPTION으로 받은 오브젝트 권한을 다른 사용자에게 부여할 수 있음
- 중간관리자(WITH GRANT OPTION으로 권한을 부여받은 자)가 부여한 권한은 중간관리자만 회수 가능
- 중간관리자에게 부여된 권한 회수 시 제 3 자에게 부여된 권한도 함께 회수됨**

### 2. WITH ADMIN OPTION

- WITH ADMIN OPTION을 통해 부여 받은 시스템 권한/를 권한을 다른 사용자에게 부여할 수 있음
- 중간관리자를 거치지 않고 직접 회수 가능
- 중간관리자 권한 회수시 제 3 자에게 부여된 권한도 함께 회수 X(남아있음)**

예제) WITH GRANT OPTION / WITH ADMIN OPTION TEST

### 권한 부여)

```
SQL1 *
1 -- SYS 계정 수행)
2 GRANT SELECT ON SCOTT.ROLLBACK_TEST TO HDALAB WITH GRANT OPTION;
3 GRANT CREATE ANY TABLE TO HDALAB WITH ADMIN OPTION;
4 GRANT ALTER ANY TABLE TO HDALAB WITH ADMIN OPTION;
5
6 -- HDALAB 계정 수행)
7 GRANT SELECT ON SCOTT.ROLLBACK_TEST TO PARK;
8 GRANT CREATE ANY TABLE TO PARK;
9 GRANT ALTER ANY TABLE TO PARK;
```

### 오브젝트 권한 회수 시도)

```
SQL1 *
1 --SYS 계정 수행)
2 ► REVOKE SELECT ON SCOTT.ROLLBACK_TEST FROM PARK; -- 직접회수불가

Result
Grid Result Server Output Text Output Explain Plan Statistics

ORA-01927: 허가하지 않은 권한을 REVOKE할 수 없습니다
```

- ☞ 중간관리자를 통해 부여한 제 3 계정의 권한은 관리자가 직접 회수 불가

```
SQL1 *
1 --SYS 계정 수행)
2 REVOKE SELECT ON SCOTT.ROLLBACK_TEST FROM PARK;      -- 직접회수불가
3 REVOKE SELECT ON SCOTT.ROLLBACK_TEST FROM HDATALAB;   -- 중간관리자회수
```

☞ 대신 중간관리자에게 부여된 권한을 회수(회수 시 제 3의 계정에 부여된 권한도 함께 회수됨)

```
SQL1 *
1 --PARK 계정 수행)
2 ▶ SELECT * FROM SCOTT.ROLLBACK_TEST;    -- 조회불가
```

Result

Grid Result Server Output Text Output Explain Plan Statistics

ORA-00942: 테이블 또는 뷰가 존재하지 않습니다

☞ 중간관리자(HDATALAB)에 의해 부여된 제 3의 계정(PARK) 권한도 함께 회수됨

### 시스템 권한 회수 시도)

```
SQL1 *
1 --SYS 계정 수행)
2 REVOKE ALTER ANY TABLE FROM PARK;      -- 직접회수가능
3 REVOKE CREATE ANY TABLE FROM HDATALAB; -- 중간관리자회수
```

☞ 제 3의 계정에 부여된 권한을 관리자가 직접 회수할 수 있음

```
SQL1 *
1 -- PARK 계정 수행)
2 ▶ CREATE TABLE CREATE_TEST(COL1 NUMBER);  -- 가능
```

Result

Grid Result Server Output Text Output Explain Plan Statistics

Statement Processed.

☞ 중간 관리자의 시스템 권한을 회수하더라도 중간관리자가 제 3의 계정에 부여한 권한은 회수되지 않아 테이블 생성이 가능함