



배워두면 좋은 SQL



자격증 준비하면서 익히자!

기본 배경 지식부터  
자격증 준비까지 썩~다



# 오늘 다룰 내용은 시험 범위 중 ...

## ■ 과목 I 데이터 모델링의 이해 (10 문항)

- 제1장 데이터 모델링의 이해
- 제2장 데이터 모델과 SQL

## ■ 과목 II SQL 기본과 활용 (40 문항)

- 제1장 SQL 기본 ◀ 여기에 해당하는 부분!
- 제2장 SQL 활용
- 제3장 SQL 최적화 기본 원리



## ✓ 데이터베이스란?

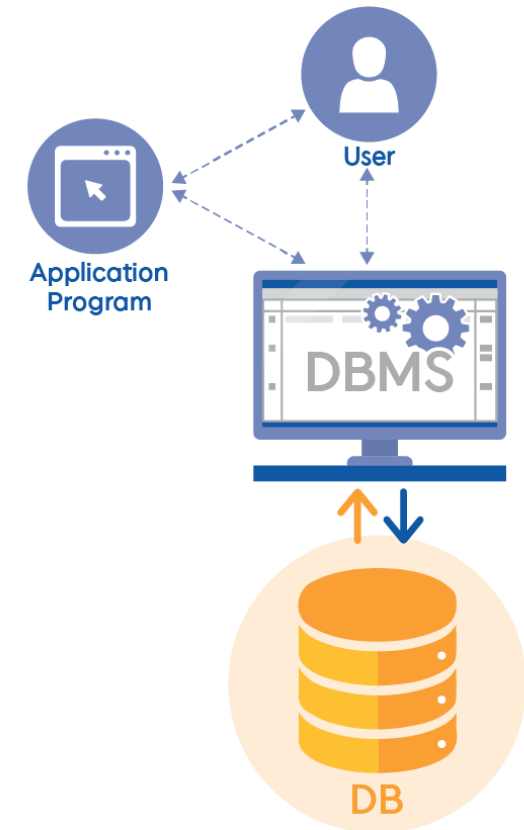
- 전자적으로 저장되고 체계적인 데이터 모음 (단어, 숫자, 이미지, 비디오 및 파일을 포함한 모든 유형의 데이터가 포함)

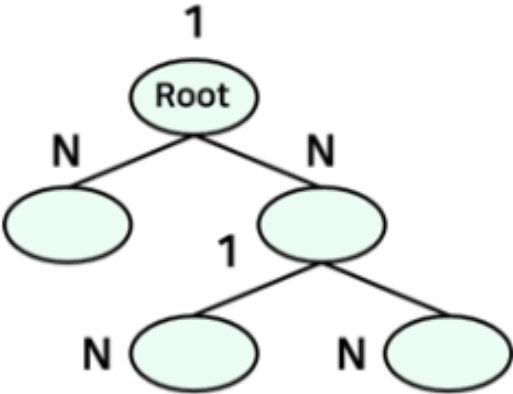
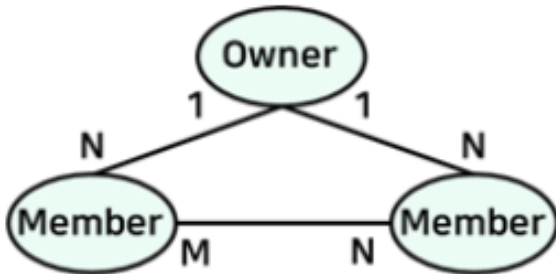
## ✓ 데이터베이스 발전 과정

- 1960년대: 파일 구조를 통한 데이터 저장 및 관리
- 1970년대: 계층형, 망형 데이터베이스 상용화
- 1980년대: **관계형 데이터베이스(Oracle, Sybase, DB2)** 상용화
- 1990년대: 객체 관계형 데이터베이스로 발전

## ✓ 관계형 데이터베이스(Relational Database)

- 1970년 E.F. Codd 박사에 의해 소개
- IBM의 SQL 개발 후 Oracle 등 여러 회사에서 상용화
- 기존 파일시스템, 계층형, 망형 DB를 대체하며 주력 DB로 자리매김



계층형 데이터베이스	네트워크 데이터베이스	관계형 데이터베이스								
<div>- 자료구조: 트리형태 (Tree) - 표현관계: 1대N 관계</div>	<div>- 자료구조: 오너-멤버 형태 - 표현관계: 1대N, M대N</div>	<div>- 자료구조: 릴레이션 (Relation) - 릴레이션으로 집합 연산, 관계 연산</div>								
		<table><tr><th>식별자</th><th>칼럼</th></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr><tr><td></td><td></td></tr></table>	식별자	칼럼						
식별자	칼럼									

## ✓ 테이블(Table)이란?

- 테이블은 관계형 데이터베이스의 핵심 구성 요소로, 데이터를 구조화하여 저장하는 2차원의 객체임.
- 각 테이블은 특정 주제나 업무에 맞추어 설계되며, 데이터의 집합을 나타냄.
- **하나의 데이터베이스는 하나 이상의 테이블을 포함할 수 있으며**, 이 테이블들은 특정한 구조로 데이터를 저장함. (**RDB의 기본 단위**)

## ✓ 정규화와 키의 중요성

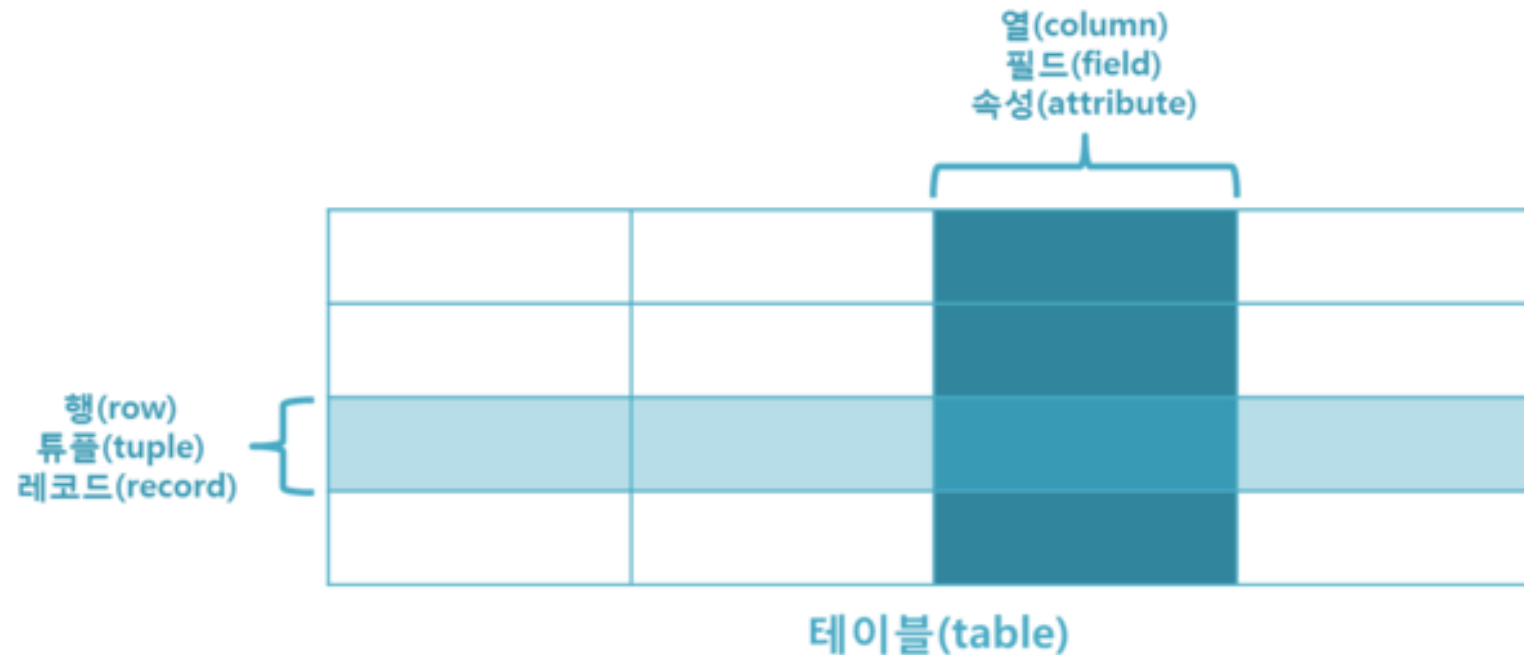
- **정규화(Normalization)**: 데이터의 중복을 최소화하고 정합성을 높이기 위한 프로세스임.
- **기본키(Primary Key)**: 각 행을 고유하게 식별할 수 있는 칼럼 또는 칼럼의 집합임. 테이블 내에서 각 레코드를 구별하는 데 사용됨.
- **외래키(Foreign Key)**: 다른 테이블의 기본키와 매칭되어 두 테이블 간의 관계를 생성하는 칼럼임.

## ✓ 테이블 간 관계

- 테이블들은 외래키를 통해 서로 연결되어 복잡한 데이터 구조와 관계를 나타낼 수 있음.
- 이러한 관계 설정은 데이터베이스의 데이터를 보다 효과적으로 관리하고, 조회할 때 높은 유연성을 제공함.

## (참고) TABLE의 구조

- **칼럼(Column):** 데이터의 종류를 정의하는 세로줄임.
  - 예를 들어 '고객 ID', '이름', '주소' 등 각각의 데이터 속성을 나타냄.
- **행(Row):** 실제 데이터가 저장되는 가로줄로, 데이터베이스의 레코드를 구성함.
  - 각 행은 테이블 내의 하나의 데이터 항목이나 사건을 나타냄.



# SQL(Structured Query Language)

## ✓ SQL이란?

- SQL(Structured Query Language)은 **관계형 데이터베이스**에서 데이터 정의, 데이터 조 작, 데이터 제어를 하기 위해 사용하는 언어

## ✓ SQL언어의 종류 (하위 집합)

### ▪ DML(Data Manipulation Language)

- 데이터를 실제로 조작하는 데 사용되는 SQL의 일부임. 데이터를 조회(**SELECT**), 추가(**INSERT**), 수정(**UPDATE**), 삭제(**DELETE**)하는 데 사용됨.

### ▪ DDL(Data Definition Language)

- 데이터베이스의 스키마를 정의하는 데 사용되는 SQL의 일부임. 테이블, 인덱스 등과 같은 데이터 구조를 생성(**CREATE**), 수정(**ALTER**), 삭제(**DROP**), 그리고 이름을 변경(**RENAME**)하는 데 사용됨.

### ▪ DCL(Data Control Language)

- 데이터베이스 사용자에게 특정 작업을 수행할 권한을 부여(**GRANT**)하거나 취소(**REVOKE**)하는 데 사용되는 SQL의 일부임.

### ▪ TCL(Transaction Control Language)

- 데이터베이스의 트랜잭션을 관리하는 데 사용되는 SQL의 일부임. 한 번에 여러 DML 명령어들을 그룹화하여 전체 단위로서 실행하거나 되돌릴 수 있게 함.
- 이를 통해 **COMMIT** 명령어로 트랜잭션을 완료하거나, **ROLLBACK** 명령어로 이전 상태로 되돌릴 수 있음.



## (참고) SQL 종류 | DDL, DML, DCL, TCL

DDL	DML	DCL	TCL
관계형 데이터베이스의 구조를 정의함	테이블에서 데이터를 입력, 수정, 삭제, 조회함	데이터베이스 사용자에게 권한을 부여, 회수함	트랜잭션을 제어하는 명령어
<ul style="list-style-type: none"><li>- CREATE</li><li>- ALTER</li><li>- DROP</li><li>- RENAME</li></ul>	<ul style="list-style-type: none"><li>- INSERT</li><li>- UPDATE</li><li>- DELETE</li><li>- SELECT</li></ul>	<ul style="list-style-type: none"><li>- GRANT</li><li>- REVOKE</li><li>- TRUNCATE</li></ul>	<ul style="list-style-type: none"><li>- COMMIT</li><li>- ROLLBACK</li><li>- SAVEPOINT</li></ul>



# **DDL**

## **(Data Definition Language)**

# DDL(Data Definition Language)

## ✓ DDL(Data Definition Language)

- DDL은 데이터베이스 스키마를 **생성, 변경, 삭제**하기 위한 언어임.
- 데이터베이스의 구조를 정의하는 SQL의 한 부분으로 테이블, 인덱스, 뷰 등 **데이터베이스 객체를 정의함**.
- DDL의 이해는 **데이터베이스 설계와 구조화의 기본을 형성**하며, 데이터의 저장 방식과 접근 방식을 결정함.

## ✓ 데이터 유형(Data Types)의 중요성

- **데이터 유형은 테이블 내 칼럼에 저장될 데이터의 종류와 크기를 정의함**.
- 잘못된 데이터 유형 사용은 에러 발생과 데이터 무결성 손상의 주요 원인이 될 수 있음.
- 데이터 유형은 데이터의 **효율적인 저장 및 검색과 직결되며, 시스템의 성능에 영향을 미칠 수 있음**.

## ✓ 대표적인 데이터 유형(**DATATYPE**) 및 기능

- **CHARACTER(s)**: 고정 길이 문자열. 정의된 길이보다 짧은 데이터는 공백으로 채워짐.
- **VARCHAR(s)**: 가변 길이 문자열. 할당된 변수 값의 바이트만큼만 사용되며, 공백이 데이터의 일부로 간주됨.
- **NUMERIC**: 정수와 실수를 포함한 숫자 데이터. 최대한의 한계값을 정의함.
- **DATETIME**: 날짜와 시간을 저장. Oracle과 SQL Server 간의 단위 차이가 존재함.

# DDL(Data Definition Language) : CREATE

## ✓ CREATE TABLE

- 새로운 테이블을 "생성"하는데 사용됩니다. 이 명령어를 사용할 때는 일반적으로 테이블에 데이터가 없는 상태에서 시작합니다.
- **DATATYPE**은 SQL에서 데이터베이스 컬럼의 유형을 정의하는 데 사용됩니다.
  - 숫자형 데이터 유형: INTEGER (또는 INT), DECIMAL (또는 NUMERIC), FLOAT, DOUBLE
  - 문자형 데이터 유형: CHAR(n), VARCHAR(n), TEXT
  - 날짜 및 시간 데이터 유형: DATE, TIME, DATETIME, TIMESTAMP
- **DEFAULT**은 컬럼에 대한 기본값을 정의하는 데 사용됩니다.
  - 리터럴 값: 문자열, 숫자, 불리언 등의 기본 데이터 유형에 직접 할당되는 값을 의미함.
    - 예: DEFAULT '문자열', DEFAULT 100, DEFAULT TRUE
  - 함수: 현재 날짜, 시간 등을 반환하는 SQL 함수를 사용할 수 있음.
    - 예: DEFAULT CURRENT\_DATE, DEFAULT CURRENT\_TIMESTAMP
  - NULL: 명시적으로 NULL을 DEFAULT 값으로 지정할 수 있음.
    - 예: DEFAULT NULL

## ✓ CTAS (Create Table As Select)

- 기존 테이블의 구조와 그 안의 데이터를 기반으로 테이블을 생성하는 경우에 사용됩니다.
- SELECT 문을 통해 기존 테이블에서 원하는 데이터를 선택한 후, 새 테이블로 생성합니다.

## ✓ (참고) DESCRIBE(DESC) : 생성된 테이블 구조를 확인이 가능합니다.

```
CREATE TABLE 테이블명 (  
    컬럼명1 DATATYPE [DEFAULT 형식],  
    컬럼명2 DATATYPE DEFAULT 20,  
    컬럼명3 DATATYPE DEFAULT NULL  
);
```

```
CREATE TABLE 새테이블명 AS  
  
SELECT *  
  
FROM 기존테이블명;
```

```
DESCRIBE 테이블명, DESC 테이블명
```

# DDL(Data Definition Language) : CREATE

## ✓ 테이블 생성 시 주의해야 할 규칙

### ■ 테이블명 규칙

- 테이블명은 의미 있는 **단수형 이름**을 사용하는 것을 권장함. 이미 존재하는 테이블과의 **이름 중복을 피해야 함**.
- 대소문자 구분 없이 일반적으로 **대문자로 생성되는 경향**이 있음.
- 벤더별로 길이 제한이 있으며, **시작은 문자로** 해야 함.

### ■ 칼럼명 및 데이터 유형

- 칼럼명은 테이블 내에서 유일해야 하며, 데이터 유형을 명확히 지정해야 함.
- 데이터 유형에 따라 크기 지정이 필요한 경우(예: **VARCHAR(255)**)가 있음.
- DATETIME과 같은 일부 데이터 유형은 크기를 지정하지 않음.

### ■ 칼럼 규칙 및 구문의 형식

- 각 칼럼은 콤마로 구분하되, 마지막 칼럼 뒤에는 콤마를 사용하지 않음.
- 생성 구문은 반드시 **세미콜론**으로 마무리해야 함.
- **A-Z, a-z, 0-9, \_, \$, #** 외의 특수문자는 사용할 수 없음.

### ■ 예약어와 데이터 표준화

- **예약어**는 테이블명이나 칼럼명으로 사용할 수 없음.
- 데이터베이스 전체에서 칼럼명을 일관성 있게 사용하는 것이 데이터 표준화에 도움이 됨.

### ■ 제약조건

- 칼럼에 대한 제약조건을 설정할 때는 **CONSTRAINT** 키워드를 사용하여 추가할 수 있음.
- **NOT NULL, UNIQUE** 등의 제약조건을 통해 데이터 무결성을 보장할 수 있음.

```
CREATE TABLE Users (  
    UserID INT,  
    Email VARCHAR(255),  
    CONSTRAINT UC_Email UNIQUE (Email),  
    CONSTRAINT NN_UserID NOT NULL (UserID)  
);
```

- **예약어(Reserved Word)**: SQL 언어의 구문에서 특별한 의미를 가지는 단어들  
**EX) SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, DROP**

# DDL(Data Definition Language) : **CREATE**

## ✓ 제약조건(Constraint)

- 데이터베이스에서 데이터의 정확성과 무결성을 유지하기 위해 테이블에 설정할 수 있는 규칙입니다.
- CREATE TABLE 문에서 제약조건을 정의하여 데이터가 삽입되거나 수정될 때 그 데이터가 특정 규칙을 준수하도록 강제할 수 있습니다.

## ✓ 제약조건의 종류

- **PRIMARY KEY (기본키):**
  - 테이블의 각 행(row)을 유일하게 식별하는 열(column)의 집합입니다.
  - 기본키로 지정된 칼럼은 **UNIQUE 값**이어야 하며(UNIQUE KEY), **NULL을 허용하지 않습니다**(NOT NULL).
- **UNIQUE KEY (고유키):**
  - **칼럼의 모든 값이 유일해야 함**을 지정하는 제약조건입니다.
  - **NULL 값은 고유성을 위반하지 않으며**, 하나의 칼럼에는 여러 NULL 값이 있을 수 있습니다.
- **FOREIGN KEY (외래키):**
  - 한 테이블의 칼럼이 **다른 테이블의 기본키를 참조**하도록 설정합니다.
  - 참조 무결성을 위해 사용되며, 외래키로 지정된 칼럼 값은 참조하는 테이블의 기본키 값에 존재해야 합니다.
- **NOT NULL:**
  - 칼럼에 NULL 값이 올 수 없음을 지정합니다. 이 제약조건이 설정된 칼럼은 **반드시 NULL 값이 아닌 값이 들어와야 합니다**.
- **CHECK:**
  - 입력할 수 있는 칼럼 값에 대해 특정 조건을 검사하도록 설정합니다.
  - CHECK 제약조건은 지정된 **조건식이 TRUE일 때만 데이터 입력이나 수정을 허용**합니다.
    - 예를 들어, 어떤 숫자 칼럼이 음수를 가질 수 없도록 하려면, CHECK (column\_name >= 0)을 사용할 수 있습니다.

# DDL(Data Definition Language) : ALTER

✓ **ALTER TABLE** : 이미 존재하는 데이터베이스 테이블의 구조를 변경하는 데 사용됩니다.

- 다양한 변경 사항을 적용할 수 있으며, 주로 **컬럼 추가**, **컬럼 삭제**, **컬럼 수정**, **컬럼명 변경**, 그리고 **제약조건의 추가 및 삭제** 등이 있습니다.
- **컬럼 추가 (ADD)** : 기존 테이블에 새로운 컬럼을 추가합니다. (컬럼 위치 지정 불가, 최근 추가된 컬럼 = 테이블 마지막 컬럼)
  - [형식] **ALTER TABLE** 테이블명 **ADD** 추가할\_컬럼명 데이터\_유형;
- **컬럼 삭제 (DROP COLUMN)** : 테이블에서 필요 없는 컬럼을 삭제할 수 있으며, 데이터가 있거나 없거나 모두 삭제 가능합니다.
  - 한 번에 하나의 컬럼만 삭제 가능하며, 컬럼 삭제 후 최소 하나 이상의 컬럼이 테이블에 존재해야 합니다. 이때, 주의할 부분은 한 번 삭제된 컬럼은 복구가 불가능합니다.
  - [형식] **ALTER TABLE** 테이블명 **DROP COLUMN** 삭제할\_컬럼명;
- **컬럼 수정 (MODIFY)** : 컬럼의 데이터 유형, 디폴트(DEFAULT) 값, NOT NULL 제약조건에 대한 변경을 포함할 수 있습니다.
  - [ORACLE 형식] **ALTER TABLE** 테이블명 **MODIFY** (컬럼명1 데이터\_유형 [DEFAULT 값] [NOT NULL], 컬럼명2 데이터\_유형 ...);
  - [SQL Server 형식] **ALTER TABLE** 테이블명 **ALTER COLUMN** (컬럼명1 데이터\_유형 [DEFAULT 값] [NOT NULL], 컬럼명2 데이터\_유형 ...);
- **컬럼명 변경 (RENAME COLUMN)** : 컬럼의 이름을 변경합니다.
  - [형식] **ALTER TABLE** 테이블명 **RENAME COLUMN** 원본\_컬럼명 **TO** 변경\_컬럼명
- **제약조건 삭제 (DROP CONSTRAINT)** : 테이블에서 특정 제약조건을 삭제합니다.
  - [형식] **ALTER TABLE** 테이블명 **DROP CONSTRAINT** 제약조건명
- **제약조건 추가 (ADD CONSTRAINT)** : 테이블에 새로운 제약조건을 추가합니다.
  - [형식] **ALTER TABLE** 테이블명 **ADD CONSTRAINT** 제약조건명 제약조건 (컬럼명)

# DDL(Data Definition Language) : **RENAME / DROP / TRUNCATE**

✓ **RENAME TABLE:** 테이블의 이름을 변경합니다.

- **Oracle:**

- [형식] RENAME 기존\_테이블명 TO 새로운\_테이블명;

- **SQL Server:**

- [형식] sp\_rename '기존\_테이블명', '새로운\_테이블명';

✓ **DROP TABLE:** 테이블을 데이터베이스에서 완전히 삭제합니다.

- **Oracle:**

- [형식] DROP TABLE 테이블명 [CASCADE CONSTRAINT];
- CASCADE CONSTRAINT 옵션은 테이블을 삭제하면서 모든 참조 제약조건도 함께 삭제합니다.

- **SQL Server:**

- [형식] DROP TABLE 테이블명;
- SQL Server에서는 CASCADE 옵션이 없으므로, 참조 제약조건을 먼저 삭제해야 합니다.

✓ **TRUNCATE TABLE :** 테이블의 모든 데이터를 삭제하지만, 테이블 구조는 유지합니다.

- 모든 SQL 데이터베이스 시스템 공통:

- [형식] TRUNCATE TABLE 테이블명;
- 테이블의 '구조'(칼럼, 데이터 유형, 제약조건 등)는 그대로 유지되면서, '데이터'(실제 테이블에 저장된 레코드)만 완전히 삭제됩니다.
- 시스템 리소스를 덜 사용하므로 대량의 데이터 삭제 시 DELETE보다 TRUNCATE를 사용하는 것이 좋습니다.
- TRUNCATE는 자동 적용되며, 일반적으로 복구가 불가능합니다.



# **DML**

## **(Data Manipulation Language)**

# DML(Data Manipulation Language) : DML / TRANSACTION(참고)

## ✓ DML(Data Manipulation Language)

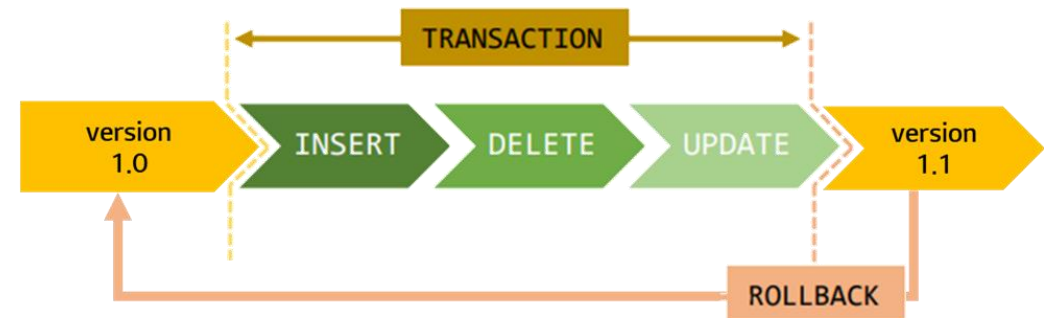
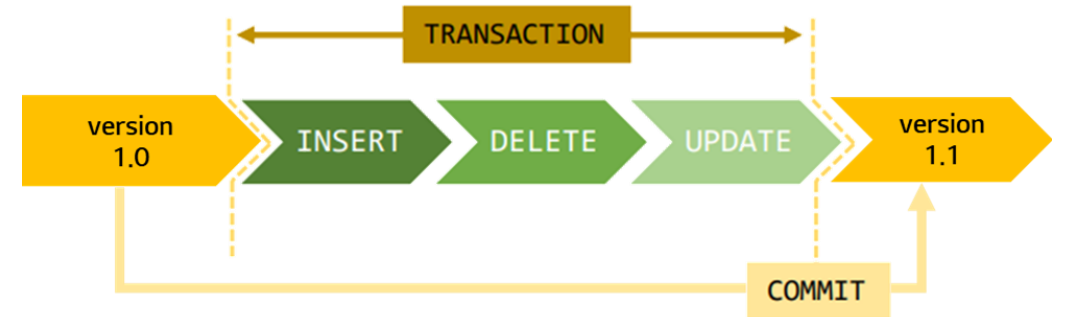
- 데이터를 조작하는 데 사용되는 SQL 명령어를 의미합니다
- 주로 **INSERT, UPDATE, DELETE, SELECT**가 있습니다.

## ✓ TRANSACTION (트랜잭션)

- DB 내에서 하나의 그룹으로 처리되어야 하는 명령문들을 모아 놓은 작업 단위입니다.

## ✓ COMMIT과 ROLLBACK

- **COMMIT** : 하나의 트랜잭션 과정을 종료하여 **변경된 모든 내용을 반영하고 저장**합니다.
  - TRANSACTION(INSERT, UPDATE, DELETE)작업 내용을 실제 DB에 저장한다.
  - 이전 데이터가 완전히 UPDATE된다.
- **ROLLBACK** : 트랜잭션으로 처리가 시작되기 **이전의 상태로 되돌립니다**.
  - TRANSACTION(INSERT, UPDATE, DELETE)작업 내용을 취소한다.
  - 이전 COMMIT한 곳까지만 복구한다.
- COMMIT과 ROLLBACK은 TCL(Transaction Control Language)에서 더 자세하게 다룰 예정임



# DML(Data Manipulation Language) : **INSERT / UPDATE / DELETE**

## ✓ 1. INSERT : 데이터를 새로 입력하는 명령어입니다. (2가지 방법 존재)

- **방법1.** 지정된 컬럼 리스트에 대응하는 값을 명시적으로 입력합니다. (입력되지 않은 컬럼은 NULL이나 DEFAULT 값이 입력됩니다)
  - [형식] **INSERT INTO** 테이블명 (컬럼1, 컬럼2, 컬럼3) **VALUES** (값1, 값2, 값3);
  - Ex. 만약 Players 테이블에 이름(name), 나이(age), 포지션(position)을 삽입하려면 => INSERT INTO Players (name, age, position) VALUES ('홍길동', 25, '미드필더');
- **방법2.** 컬럼 리스트 명시하지 않고, 모든 컬럼에 대한 값을 순차적으로 입력합니다. (컬럼의 순서대로 빠짐없이 데이터가 입력되어야 합니다.)
  - [형식] **INSERT INTO** 테이블명 **VALUES** (값1, 값2, 값3, ...);
  - Ex. Players 테이블에 모든 컬럼에 대한 값을 삽입하려면: => INSERT INTO Players VALUES ('홍길동', 25, '미드필더');

## ✓ 2. UPDATE

- 입력한 정보 중에 잘못 입력되거나 변경이 발생하여 정보를 수정해야 하는 경우 사용합니다.
  - [형식] **UPDATE** 테이블명 **SET** 수정할\_컬럼1 = 수정할\_값1, 수정할\_컬럼2 = 수정할\_값2 [WHERE 수정조건];
  - Ex. Players 테이블에서 id가 10인 선수의 이름과 나이를 업데이트하려면: => UPDATE Players SET name = '김철수', age = 30 WHERE id = 10;

## ✓ 3. DELETE

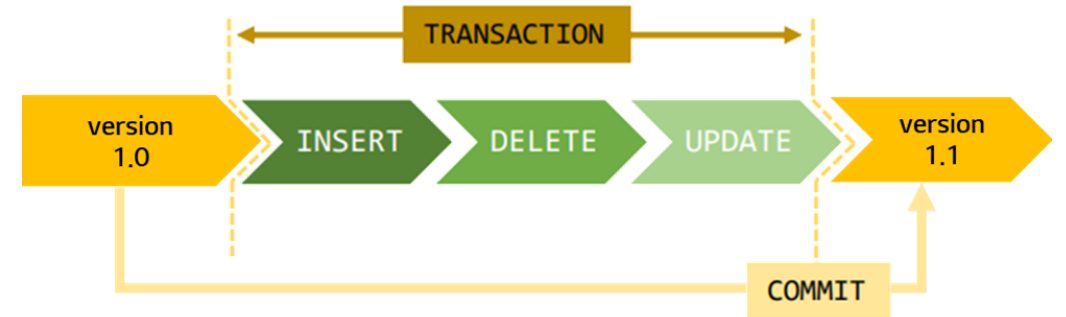
- 테이블에서 특정 조건을 만족하는 데이터를 삭제합니다.
  - [형식] **DELETE FROM** 테이블명 **WHERE** 조건;
  - Ex. Players 테이블에서 나이가 30세 이상인 선수들을 삭제하려면: => DELETE FROM Players WHERE age >= 30;

# DML(Data Manipulation Language) : SQL 제공자 별 Transaction

## ✓ DML 작업과 트랜잭션 관리

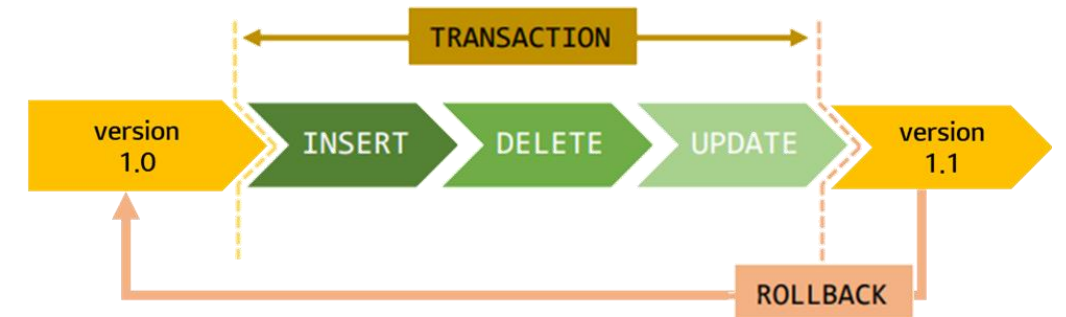
### ▪ Oracle

- 트랜잭션 처리:
  - DML 작업(INSERT, UPDATE, DELETE)은 메모리 버퍼에 임시 저장됨.
  - 변경 사항은 COMMIT 명령 실행 시 데이터베이스에 영구적으로 반영됨.
  - ROLLBACK 명령을 통해 변경 사항을 취소할 수 있음.
- 변경 확인:
  - COMMIT 전에는 다른 세션에서 변경 사항을 볼 수 없음.
  - 변경 사항은 트랜잭션 내에서만 확인 가능.



### ▪ SQL Server

- AUTO COMMIT 기본 설정:
  - 각 DML 명령 후 자동으로 COMMIT됨.
  - 명시적인 BEGIN TRANSACTION을 사용해야 수동 트랜잭션 관리 가능.
- 트랜잭션 관리:
  - COMMIT 또는 ROLLBACK을 사용하여 수동으로 트랜잭션을 종료할 수 있음.



# (참고) DML DELETE vs DDL TRUNCATE

✓ DELETE와 TRUNCATE는 **데이터베이스에서 데이터를 제거**하는 두 가지 기본적인 방법입니다. 이 둘을 명백하게 구분하실 필요가 있습니다.

## ✓ DELETE

- **분류:** DELETE는 **DML(Data Manipulation Language)**에 속합니다. DML은 데이터를 조작하는 데 사용되며, 데이터베이스의 구조를 변경하지 않고 데이터 내용만을 수정하거나 삭제합니다.
- **작동 방식:** DELETE 명령은 테이블에서 하나 이상의 행을 조건에 따라 삭제합니다. 이 때, 각 행의 삭제는 **트랜잭션 로그에 기록**됩니다.
- **롤백 가능:** 로그에 기록된 정보를 통해, **DELETE로 삭제된 데이터는 롤백 명령을 통해 복구**할 수 있습니다. 이는 트랜잭션 내에서 데이터의 무결성을 보장하며, 오류가 발생했을 때 원상태로 돌릴 수 있는 안전장치를 제공합니다.
- **시스템 부하:** DELETE는 각 행을 개별적으로 처리하고, 각각의 삭제 연산에 대해 로그를 생성하기 때문에, 대량의 데이터를 삭제할 때 **시스템에 상대적으로 높은 부하**를 줄 수 있습니다.

## ✓ TRUNCATE

- **분류:** TRUNCATE는 **DDL(Data Definition Language)**의 요소로 분류될 수 있습니다. 비록 테이블의 데이터를 삭제하는 것이 주된 기능이지만, 이 명령은 테이블의 데이터 페이지를 직접 해제하고 초기화하는 저-수준 작업을 수행하기 때문에 DDL의 특성을 갖습니다.
- **작동 방식:** TRUNCATE는 테이블의 **모든 행을 빠르게 삭제**하고, 데이터 저장에 사용되었던 공간을 재사용 가능하게 만듭니다. 이 과정에서 특정 데이터 페이지를 직접 해제합니다.
- **롤백 불가능:** 일반적인 상황에서 TRUNCATE은 트랜잭션 로그를 (거의) 생성하지 않기 때문에 **ROLLBACK이 불가능**합니다.
- **시스템 부하:** TRUNCATE는 전체 테이블의 데이터를 **일괄 삭제**하는 방식으로 작동하기 때문에, 처리 속도가 매우 빠르며 **시스템 부하도 적습니다**.

# DML(Data Manipulation Language) : **SELECT**

## ✓4. **SELECT** : 테이블에서 데이터를 조회합니다 (기본 형식 우측 참고)

- **ALL** : (Default) 옵션 중복된 데이터가 있어도 모두 출력

- **DISTINCT** : 중복 데이터 제거

- [형식] **SELECT DISTINCT** 컬럼명 **FROM** 테이블명;

- Ex. Players 테이블에서 중복 없이 국가명만 조회하려면: => SELECT DISTINCT country FROM Players;

- **ALIAS** : 별칭 지정

- [형식] **SELECT** 컬럼명 **AS** '별칭' **FROM** 테이블명;

- Ex. Players 테이블에서 선수의 이름(name)을 'Player Name'으로 표시하려면: => SELECT name AS 'Player Name' FROM Players;

- **WILDCARD** : 모든 컬럼 조회

- [형식] **SELECT \* FROM** 테이블명;

- Ex. Players 테이블의 모든 데이터를 조회하려면: => SELECT \* FROM Players;

- **ORDER BY** : 데이터를 오름차순(ASC, 기본값) or 내림차순(DESC)으로 정렬하여 출력

- 정렬 시점: 모든 실행이 끝난 후, 데이터를 출력하기 바로 전 시점

- [형식] **SELECT DISTINCT** 컬럼명 **FROM** 테이블명 **ORDER BY** 컬럼명 **DESC**;

- Ex. 고객 테이블에서 고객의 나이를 오름차순으로 정렬하여 조회 => SELECT DISTINCT age FROM Customers ORDER BY age ASC;

```
SELECT [ALL/DISTINCT] 보고 싶은 컬럼명, 보고 싶은 컬럼명, ...
```

```
FROM 해당 컬럼들이 있는 테이블명;
```

# DML(Data Manipulation Language) : 연산자

## ✓ 산술연산자

- +, -, \*, / (사칙연산) 등을 사용하여 컬럼 간 수학적 계산을 수행합니다.
- Label name이 길어지기에 ALIAS로 별칭을 사용하는 것을 권장합니다.
- 수학과 마찬가지로 (), \*, /, +, - 의 우선순위를 가진다.

```
SELECT PLAYER_NAME AS 이름, HEIGHT - WEIGHT AS "키-몸무게"  
FROM PLAYER;
```

## ✓ 합성연산자

- 문자열을 연결합니다.
  - Oracle을 사용하는 경우, 2개의 수직 바(||)를 사용하여 연결한다
  - SQL Server를 사용하는 경우, “+ “ 연산자를 사용하여 연결한다.
  - 두 벤더 모두 공통적으로 CONCAT(string1, string2) 함수를 사용할 수 있다.
- 칼럼과 문자 또는 다른 칼럼과 연결시킨다.
- 문자 표현식의 결과에 의해 새로운 칼럼을 생성한다.

```
-- Oracle  
SELECT PLAYER_NAME || ' 선수'  
FROM PLAYER;  
  
-- SQL Server  
SELECT PLAYER_NAME + ' 선수'  
FROM PLAYER;
```



# **TCL**

## **(Transaction Control Language)**

# TCL(Transaction Control Language)

## ✓ Transaction이란?

- 데이터베이스의 **논리적으로 연산 단위**로, 밀접히 관련되어 **분리할 수 없는 한 개 이상의 데이터 베이스 조작**을 의미합니다.
- 분할할 수 없는 최소 단위이기 때문에, 모든 연산이 성공적으로 완료되거나, 하나라도 실패할 경우 작업 전체를 원래 상태로 되돌리는 **‘모두 또는 전혀 아님(ALL or NOTHING)’** 특징이 있습니다.
- UPDATE, INSERT, DELETE, SELECT과 같은 **DML문은 트랜잭션의 대상**입니다.
- TCL은 ACID라는 특성을 가지고 있습니다 (아래 참고)

## ✓ TCL의 특성 (ACID) :

- 원자성(**A**tomicity): 트랜잭션의 모든 연산이 **완전히 수행되거나 전혀 수행되지 않아야 함**.
- 일관성(**C**onsistency): 트랜잭션이 **성공적으로 완료되면, 데이터베이스는 항상 일관된 상태**를 유지함.
- 고립성(**I**solation): 동시에 실행되는 여러 트랜잭션이 **서로 간섭 없이 독립적으로 실행**되어야 함.
- 지속성(**D**urability): 트랜잭션이 성공적으로 완료되면, 그 **결과는 영구적으로 데이터베이스에 저장**됨.

# TCL(Transaction Control Language) : COMMIT

✓ COMMIT : 수정된 모든 데이터를 데이터베이스에 영구적으로 저장함.

## ▪ COMMIT 실행 전의 데이터 상태

- 현재 사용자는 SELECT 문을 통해 변경된 데이터를 조회할 수 있으며, 아직 COMMIT되지 않았기 때문에 메모리 내에만 존재합니다.
- 기타 사용자는 현재 사용자의 변경 사항을 볼 수 없습니다. 이는 데이터의 고립성을 유지하고, 확정되지 않은 정보의 유출을 방지합니다.

## ▪ 실행 코드

```
UPDATE PLAYER SET HEIGHT = 174 WHERE NAME = '의석';  
COMMIT;
```

## ▪ COMMIT 실행 후의 데이터 상태

- COMMIT 명령을 실행하면 모든 변경 사항이 데이터베이스의 디스크에 저장되어 영구적으로 반영됩니다.
- 관련된 모든 데이터의 잠금이 해제되어, 다른 사용자들이 해당 행을 자유롭게 조작할 수 있게 됩니다.

## ▪ SQL Server에서의 COMMIT 처리

- SQL Server는 기본적으로 **AUTO COMMIT 모드**를 사용합니다.
  - 이는 사용자가 별도로 COMMIT이나 ROLLBACK을 지정하지 않아도 각 SQL 명령마다 **자동으로 COMMIT이 수행**됩니다.
- 직접 TRANSACTION을 지정해주고 싶으면 명시적으로 **BEGIN TRANSACTION, COMMIT TRANSACTION**을 입력해주면 됩니다.

# TCL(Transaction Control Language) : **ROLLBACK**

✓ ROLLBACK : 트랜잭션의 변경 사항을 취소하고 이전 상태로 복원함

- 실행 코드

```
DELETE FROM PLAYER;  
ROLLBACK;
```

- ROLLBACK 후의 데이터 상태

- 데이터 변경의 취소: ROLLBACK을 실행하면 해당 트랜잭션 내에서 이루어진 모든 데이터 변경 작업이 취소됩니다.
- 이전 데이터의 복원: 삭제된 레코드, 변경된 필드, 추가된 행 등 모든 데이터가 트랜잭션 시작 전의 상태로 복구됩니다.
- 잠금 해제: 트랜잭션 도중 설정된 모든 데이터 잠금이 해제되어, 다른 사용자들이 해당 데이터를 다시 자유롭게 조작할 수 있게 됩니다.

- SQL Server의 ROLLBACK

- 기본적으로 AUTO COMMIT 모드를 사용하므로, ROLLBACK을 수행하려면 명시적으로 트랜잭션을 선언해야 합니다.

```
BEGIN TRANSACTION  
DELETE FROM PLAYER;  
ROLLBACK;
```

# TCL(Transaction Control Language) : **SAVEPOINT**

✓ **SAVEPOINT** : 트랜잭션 내에서 특정 지점을 기록(마킹) 함.

(롤백 시, 전체 롤백이 아닌 특정 지점까지만 부분적으로 롤백이 가능합니다)

- EX. '**ROLLBACK TO A**'를 수행하면 '**UPDATE**'와 '**DELETE**' 연산 이후의 변경 사항이 취소되고, '**INSERT**' 연산 직후의 상태로 되돌아갑니다.

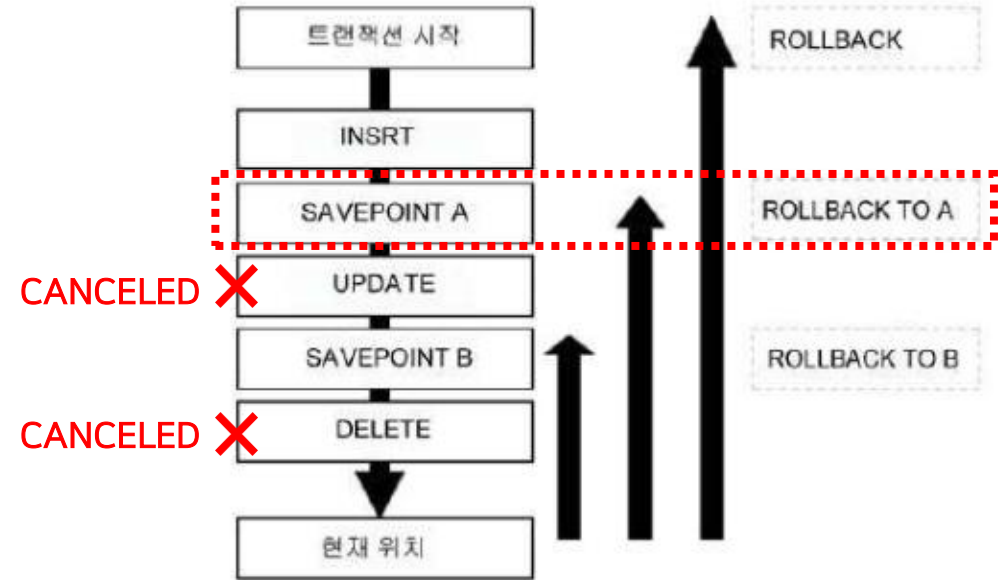
## ■ 저장점의 관리:

- 복수의 **SAVEPOINT**를 설정할 수 있으며, 같은 이름으로 여러 번 **SAVEPOINT**를 설정할 경우 **가장 마지막에 설정된 SAVEPOINT가 유효**합니다.
- 특정 **SAVEPOINT**로 롤백을 수행하면 **그 이후에 설정된 모든 SAVEPOINT는 무효**됩니다.  
예를 들어, '**ROLLBACK TO A**' 이후에는 '**SAVEPOINT B**'가 더 이상 유효하지 않습니다.

## ■ 실행 코드

```
-- Oracle  
SAVEPOINT SVPT1;  
ROLLBACK TO SVPT1;
```

```
-- SQL Server  
SAVE TRANSACTION SVTR1;  
ROLLBACK TRANSACTION SVTR1;
```



# TCL(Transaction Control Language) : ETC

## ✓ COMMIT과 ROLLBACK의 효과

- 데이터 무결성 보장
  - 데이터베이스의 무결성을 보호하고, 실수나 오류로 인한 부정확한 데이터 입력을 방지합니다.
- 영구적인 변경 이전의 사전 확인
  - ROLLBACK 명령을 사용함으로써, 영구적인 변경을 적용하기 전에 이러한 변경들이 올바른지 확인할 수 있습니다.
- 논리적으로 연관된 작업의 그룹핑
  - 여러 데이터 변경 작업을 하나의 트랜잭션으로 그룹화하여 관리함으로써, 관련 작업들이 일관되게 처리되도록 합니다.

## ✓ (ORACLE) COMMIT과 ROLLBACK 없이 트랜잭션이 종료되는 경우:

- DDL(Data Definition Language, CREATE/ALTER/DROP/TRUNCATE 등) 명령어는 실행 즉시 자동 COMMIT 됨.
  - 이는 테이블 정의(구조) 자체에 대한 생성이나 수정 사항이기 때문에 실행하게 되면 자동으로 COMMIT 됨.
- 반면에 DML(Data Manipulation Language, INSERT/UPDATE/DELETE 등) 명령어는 실행 즉시 자동 COMMIT 되지 않음
  - 이때 DML 실행 후에 DDL 명령어가 실행되면, DDL 명령어로 인해 DML 내용까지 자동으로 COMMIT 됨.
  - COMMIT없이 비정상적인 종료 시에는 트랜잭션이 ROLLBACK되게 됨.

# SQL문법: WHERE



# WHERE

## ✓ WHERE 절 개요

- WHERE 절은 데이터베이스에서 특정 조건에 맞는 데이터만 조회하기 위해 사용됨.
- 필요 없는 데이터를 불러오지 않아 시스템 자원을 효율적으로 사용할 수 있게 해줌.
  - 기본 구조는 다음과 같음

```
SELECT (DISTINCT/ALL) 컬럼명 (ALIAS명)
FROM 테이블명
WHERE 컬럼명 연산자 (문자/숫자/표현식 or 비교 컬럼명);
```

## ✓ 연산자의 종류

- 비교 연산자: **=, >, ≥, <, ≤** → Ex. WHERE age >= 18
- SQL 연산자:
  - BETWEEN a AND b**: a와 b 사이의 값. → Ex. WHERE age BETWEEN 18 AND 30
  - IN (list)**: 리스트 내의 값 중 하나. → Ex. WHERE department IN ('Sales', 'Engineering')
  - LIKE '패턴'**: 문자 패턴 매칭 (와일드카드 **%**, **\_** 사용). → Ex. WHERE name LIKE '김%'
  - IS NULL**: NULL 값. → Ex. WHERE address IS NULL
- 논리 연산자: **AND, OR, NOT** → Ex. WHERE age >= 18 AND age <= 30
- 부정 연산자: **^=, !=, <>, NOT EQUAL (\*같지 않다)** → Ex. WHERE age <> 25

(참고) WILD CARD 추가 설명

SQL의 LIKE 연산에서는 %와 \_만이 와일드카드로 기능합니다:

- ✓ % : 0개 이상의 문자를 대체합니다.
- ✓ \_ : 정확히 하나의 문자를 대체합니다.

(참고) 연산자 우선 순위

연산 우선순위	설 명
1	괄호 ()
2	NOT 연산자
3	비교 연산자, SQL 비교 연산자
4	AND
5	OR

# ROWNUM / TOP

## ✓ ROWNUM (Oracle)

- Oracle에서 ROWNUM은 쿼리 결과 집합에서 각 행에 부여되는 일련번호를 나타내는 칼럼입니다. (\*원하는 행의 수만큼 제한할 때 사용)
- 기본 사용법:
  - 한 건의 데이터만 추출할 때: WHERE ROWNUM = 1
  - N건 이하의 데이터를 추출할 때: WHERE ROWNUM <= N
- 특징:
  - 첫 번째 조건을 충족하는 N건의 데이터를 정확히 가져오려면, 서브쿼리와 함께 사용해야 함.
  - ROWNUM은 쿼리가 실행되면서 부여되므로, 직접적인 ROWNUM 비교는 그 한계를 갖습니다.

```
SELECT PLAYER_NAME FROM PLAYER WHERE ROWNUM = 1;
```

```
SELECT PLAYER_NAME FROM PLAYER WHERE ROWNUM <= 5;
```

## ✓ TOP 절 (SQL Server)

- SQL Server에서 TOP 절은 결과 집합으로부터 지정된 수의 행 또는 특정 퍼센트의 행만 반환할 목적으로 사용됩니다.
- 기본 사용법: **TOP (Expression) [PERCENT] [WITH TIES]**
  - **Expression**: 반환할 행의 수 또는 퍼센트를 지정합니다.
  - **PERCENT**: 전체 결과 집합 중 상위 Expression%만큼의 행을 반환합니다.
  - **WITH TIES**: ORDER BY 절이 지정된 경우에만 사용할 수 있으며, 마지막 행과 같은 값이 있는 경우 추가 행이 출력되도록 합니다

```
SELECT TOP(10) PERCENT PLAYER_NAME FROM PLAYER ORDER BY SCORE;
```

```
SELECT TOP(5) WITH TIES PLAYER_NAME FROM PLAYER ORDER BY HEIGHT;
```

# SQL문법: 함수(FUNCTION)

# SQL 함수 분류(유형)

✓ **SQL 함수**는 데이터베이스 내의 데이터를 조회, 변환, 계산, 비교하는 데 사용되는 미리 정의된 연산들로, 데이터를 처리하는 SQL 쿼리의 효율성과 간결성을 높이기 위해 설계되었음. SQL함수는 크게 아래와 같이 구분됨:

- **단일행 함수 (Single-Row Functions):**

- 각각의 데이터 행에 대해 독립적으로 작용하며, 각 행마다 하나의 결과를 반환함.
- 주로 문자열 처리, 수치 계산 등의 기능을 수행함.

- **다중행 함수 (Multi-Row Functions):**

- 여러 행의 데이터를 기반으로 하여 단일 결과를 생성함.
- 집계 함수와 윈도우 함수가 이 범주에 속함.

- **변환형 함수:**

- 데이터의 형태나 유형을 변환함.
- 이는 데이터의 포맷을 변경하거나, 데이터 유형을 다른 유형으로 캐스팅할 때 사용됨.

- **NULL 처리 함수:**

- 데이터베이스 내에서 NULL 값을 다루는 특수한 함수로, NULL 값을 기타 값으로 대체하거나 NULL 값을 처리하는 데 사용됨.

- **조건 표현식:**

- 주어진 조건에 따라 다른 결과를 반환하는 논리적 표현을 가능하게 함. 이를 통해 복잡한 데이터 조작을 단순화함.

# 단일행 함수 (Single-Row Functions)

✓ 각 행별로 독립적으로 작용하며 하나의 결과를 반환함.

## ■ 문자형 처리 함수:

- LOWER('SQL Expert') → 'sql expert': 문자열을 소문자로 변환함.
- UPPER('SQL Expert') → 'SQL EXPERT': 문자열을 대문자로 변환함.
- SUBSTR('SQL Expert', 5, 3) → 'Exp': 문자열의 특정 부분을 추출함.
- LENGTH('SQL Expert') → 10: 문자열의 길이를 반환함.
- ASCII('A') → 65: 문자의 ASCII 코드를 반환함.
- CHR(65) → 'A': ASCII 코드에 해당하는 문자를 반환함.
- RTRIM('http://example.com/', '/') → 'http://example.com': 특정 문자열의 오른쪽 끝에서 원하는 문자를 제거
  - 아무것도 입력하지 않을 시, RTRIM('SQL Expert ') → 'SQL Expert': 문자열 오른쪽 끝의 공백을 제거함.
- LTRIM('///http://example.com/', '/') → 'http://example.com/': 특정 문자열의 왼쪽 끝에서 원하는 문자를 제거
  - 아무것도 입력하지 않을 시, LTRIM(' SQL Expert') → 'SQL Expert': 문자열 왼쪽 끝의 공백을 제거함.

## ■ 수치형 처리 함수:

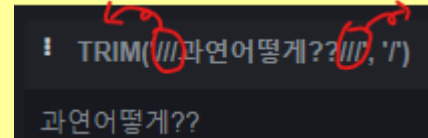
- ABS(-15) → 15: 절대값 값을 반환함.
- ROUND(38.5235, 1) → 38.5: 주어진 소수점 자리에서 반올림을 수행함.
- CEIL(38.123) → 39: 주어진 실수 값보다 크거나 같은 가장 작은 정수로 올림함.
- FLOOR(38.123) → 38: 주어진 실수 값보다 작거나 같은 가장 큰 정수로 내림함.
- MOD(7, 3) → 1: 나눗셈 수행 후 나머지 값을 반환함.

Q. 그럼 TRIM은요?

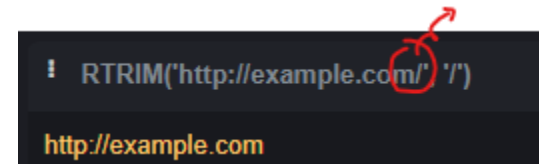
TRIM('///과연어떻게??///', '/')

→ '과연어떻게??'

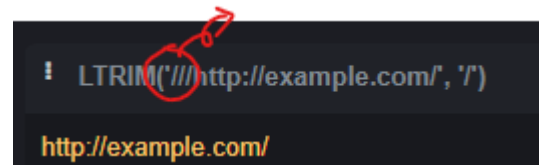
\*양쪽 끝에 원하는 문자(/)가 제거됨



! TRIM('///과연어떻게??///', '/')  
과연어떻게??



! RTRIM('http://example.com/', '/')  
http://example.com



! LTRIM('///http://example.com/', '/')  
http://example.com/

# 다중행 함수 (Multi-Row Functions)

✓ 여러 행의 데이터를 기반으로 한 결과를 생성함.

## ▪ 집계 함수:

- SUM(column\_name): 지정된 열의 총합을 계산함.
- AVG(column\_name): 지정된 열의 평균값을 계산함.
- COUNT(column\_name): 지정된 열의 행 수를 계산함.
- MAX(column\_name): 지정된 열의 최대값을 찾음.
- MIN(column\_name): 지정된 열의 최소값을 찾음.

## ▪ 윈도우 함수:

- ROW\_NUMBER() OVER (ORDER BY column\_name): 행 번호를 순서대로 할당함.
- RANK() OVER (ORDER BY column\_name): 특정 조건에 따른 순위를 매김.

# 형변환 / NULL처리 / 조건표현식 함수

## ✓ 형변환 함수

- 데이터의 유형을 변환하거나 데이터 형식을 조작함.
  - TO\_CHAR(number\_or\_date, 'format'): 숫자나 날짜를 문자열로 변환함.
  - TO\_NUMBER('string', 'format'): 문자열을 숫자로 변환함.
  - TO\_DATE('string', 'format'): 문자열을 날짜로 변환함.
  - CAST(expression AS datatype): 데이터 유형을 변경함 (Ex. 문자열 → 날짜, 정수형 → 실수형 등)

## ✓ NULL 처리 함수

- NULL 값을 다루기 위한 함수임.
  - NVL(expression, replacement): 표현식이 NULL일 경우 대체값을 반환함.
  - COALESCE(expr1, expr2, ...): 여러 표현식 중 첫 번째 NULL이 아닌 값을 반환함.
  - NULLIF(expr1, expr2): 두 표현식이 같으면 NULL을 반환하고, 다르면 첫 번째 표현식을 반환함.

## ✓ 조건 표현식

- 조건에 따라 다른 결과를 반환함.
  - CASE WHEN condition THEN result [ELSE result] END: 조건에 따라 다른 결과를 반환함.

### 명시적(Explicit) 형변환

- 형변환함수를 사용해서 데이터 타입을 일치시키는 것으로 개발자가 SQL을 사용할 때 형변환 함수를 사용해야 한다.

### 암시적(Implicit) 형변환

- 개발자가 형변환을 하지 않은 경우 데이터베이스 관리 시스템이 자동으로 형변환되는 것을 의미한다..

```
SELECT CAST(25 AS FLOAT) AS FloatResult;  
  
SELECT CAST('2023-04-15' AS DATE) AS DateResult;  
  
SELECT CAST(123.456 AS INTEGER) AS IntegerResult;
```

### (정의) NULL

- ✓ NULL(ASCII 코드 00번)은 공백(BLANK, ASCII 코드 32번)이나 숫자 0(ZERO, ASCII 48)과는 전혀 다른 값이며, 조건에 맞는 데이터가 없을 때의 공집합과도 다르다.
- ✓ 'NULL'은 '아직 정의되지 않은 미지의 값'이거나 '현재 데이터를 입력하지 못하는 경우'를 의미한다.



# SQL문법: GROUP BY, HAVING 절

# 집계 함수 (Aggregate Function)

## ✓ 정의

- 여러 행들의 그룹을 단일 결과로 요약하는 함수임.
- GROUP BY 절은 행들을 소그룹화하며, SELECT, HAVING, ORDER BY 절에서 사용 가능함.

## ✓ 주요 함수:

- **COUNT(\*)**: NULL을 포함한 행의 수를 출력함.
- **COUNT(표현)**: NULL을 제외한 행의 수를 출력함.
- **SUM([DISTINCT | ALL] 표현식)**: NULL을 제외한 합계를 출력함.
- **AVG([DISTINCT | ALL] 표현식)**: NULL을 제외한 평균을 출력함.
- **MAX([DISTINCT | ALL] 표현식), MIN([DISTINCT | ALL] 표현식)**: 최대값과 최소값을 출력함.
- **STDDEV([DISTINCT | ALL] 표현식), VARIAN([DISTINCT | ALL] 표현식)**: 표준 편차와 분산을 출력함.

## ✓ 예시

```
SELECT COUNT(*), MAX(HEIGHT), MIN(HEIGHT), ROUND(AVG(HEIGHT), 2)
FROM PLAYER;
```

# GROUP BY절 & HAVING절

## ✓ GROUP BY 절

- SQL 문에서 FROM 절과 WHERE 절 다음에 오며, 데이터를 작은 그룹으로 분류하여 그룹별 통계 정보를 얻는데 사용됨.
- 집계 함수와 함께 사용되어 그룹별 요약 정보를 제공함.
- ALIAS 명을 사용할 수 없으며, WHERE 절이 집계 전 데이터 필터링을 담당함.

```
SELECT POSITION, AVG(HEIGHT)
FROM PLAYER
GROUP BY POSITION;
```

## ✓ HAVING 절

- HAVING 절은 **GROUP BY 절에 의해 생성된 그룹의 결과에 조건을 적용**할 때 사용됨.
- WHERE 절과 유사하게 조건을 제공하지만, 그룹화된 결과에 대해 작동함.
- 집계 함수와 함께 사용되어 특정 조건을 만족하는 그룹만을 필터링함.

```
SELECT POSITION, ROUND(AVG(HEIGHT), 2)
FROM PLAYER
GROUP BY POSITION
HAVING AVG(HEIGHT) >= 180;
```

# CASE 표현을 활용한 월별 데이터 집계

- ✓ CASE 문을 사용하여 특정 조건에 따라 다른 값을 선택하며, 이를 GROUP BY 절과 함께 사용하여 월별 데이터를 집계할 수 있음.
- ✓ CASE 표현식을 통해 각 월별 급여 데이터를 분리하여 처리하고, 이후 AVG 함수 등으로 평균을 계산함.

```
-- 부서 번호별로 각 월의 평균 급여를 계산합니다.  
SELECT DEPTNO,    -- 부서 번호를 선택합니다.  
  
    -- 1월에 해당하는 직원들의 급여의 평균을 계산합니다.  
    AVG(CASE MONTH WHEN 1 THEN SAL END) M01,  
  
    -- 2월에 해당하는 직원들의 급여의 평균을 계산합니다.  
    AVG(CASE MONTH WHEN 2 THEN SAL V) M02,  
  
    -- 이하 동일한 패턴으로 3월부터 12월까지 계속됩니다.  
    AVG(CASE MONTH WHEN 3 THEN SAL END) M03,  
(생략)  
  
-- EMP 테이블에서 부서 번호, 고용 날짜에서 추출한 월, 급여 정보를 선택하는 서브쿼리입니다.  
FROM (SELECT DEPTNO, EXTRACT(MONTH FROM HIREDATE) MONTH, SAL FROM EMP)  
GROUP BY DEPTNO; -- 결과를 부서 번호(DEPTNO)별로 그룹화합니다.
```

# 만약 NULL값이 있다면?

## ✓ 집계 함수와 NULL

- 집계 함수는 NULL 값을 입력으로 받았을 때, 그 값들을 제외하고 계산을 수행함.
- CASE 표현식에서 ELSE 절을 생략하면 기본값으로 NULL이 할당됨.
- **NVL(Non-Value Logic) 함수**를 사용하여 NULL 결과를 다룰 수 있으며, 이는 보고서의 가독성을 높이는 데 유용함.

```
SELECT DEPTNO,  
       NVL(AVG(CASE MONTH WHEN 1 THEN SAL END), 0) M01,  
       NVL(AVG(CASE MONTH WHEN 2 THEN SAL END), 0) M02,  
       NVL(AVG(CASE MONTH WHEN 3 THEN SAL END), 0) M03,  
       NVL(AVG(CASE MONTH WHEN 4 THEN SAL END), 0) M04,  
       NVL(AVG(CASE MONTH WHEN 5 THEN SAL END), 0) M05,  
       NVL(AVG(CASE MONTH WHEN 6 THEN SAL END), 0) M06,  
       NVL(AVG(CASE MONTH WHEN 7 THEN SAL END), 0) M07,  
       (생략)  
  
FROM (SELECT DEPTNO, EXTRACT(MONTH FROM HIREDATE) MONTH, SAL FROM EMP)  
GROUP BY DEPTNO;
```

# SQL문법: ORDER BY 절

## ✓ 기본 정렬

- 기본적으로 ORDER BY는 오름차순(ASC)으로 정렬한다.
- 내림차순으로 정렬하고 싶다면 DESC 키워드를 사용한다.

## ✓ 칼럼명 대신 숫자 사용

- SELECT 절에 나열된 칼럼의 순서에 따라 정수를 사용하여 ORDER BY에서 참조할 수 있다.
- 이는 SQL 문장이 복잡할 때 유용하나, 유지보수와 가독성 측면에서는 권장되지 않음.

## ✓ NULL 값의 처리

- 다른 데이터베이스 시스템에서 NULL 처리가 다르다.
  - 예를 들어, Oracle은 NULL을 가장 큰 값으로 간주해 오름차순에서는 마지막에 위치시키고, 내림차순에서는 첫 번째에 위치시킴.

## ✓ SELECT 문 실행 순서와 ORDER BY의 위치

- FROM → WHERE → GROUP BY → HAVING → SELECT 순으로 데이터를 처리하고, **마지막에 ORDER BY로 정렬함.**

5. SELECT 칼럼명 [ALIAS명]  
1. FROM 테이블명  
2. WHERE 조건식  
3. GROUP BY 칼럼(Column)이나 표현식  
4. HAVING 그룹조건식  
6. ORDER BY 칼럼(Column)이나 표현식;

# (복습) 상위 N개를 가지고 오고 싶을 때

## ✓ Oracle : ROWNUM

- Oracle 데이터베이스에서는 ROWNUM이라는 시스템 칼럼을 사용하여 특정 수의 행을 추출할 수 있습니다.
- 그러나 ROWNUM은 데이터가 데이터베이스에서 어떻게 추출되었는지에 따라 값이 결정되므로, ORDER BY 절을 사용하기 전에 ROWNUM을 사용하면 정렬되지 않은 상태에서 행이 선택됩니다. 따라서, 정확한 순서로 TOP N 결과를 얻으려면 먼저 데이터를 정렬한 후 ROWNUM을 적용해야 합니다.

```
-- Oracle에서 급여가 높은 상위 3명 추출
SELECT ENAME, SAL
FROM (SELECT ENAME, SAL
      FROM EMP
      ORDER BY SAL DESC
     ) WHERE ROWNUM < 4;
```

## ✓ SQL Server : TOP N

- SQL Server에서는 TOP 키워드를 사용하여 쿼리 결과의 상위 N개의 행을 직접 선택할 수 있습니다.
- WITH TIES 옵션은 ORDER BY 절의 조건 기준으로 TOP N의 마지막 행으로 표시되는 추가 행의 데이터가 같을 경우, 해당 데이터 또한 함께 반환해줍니다.

```
-- SQL Server에서 급여가 높은 상위 2명 추출, 동점자 포함
SELECT TOP(2) WITH TIES ENAME, SAL
FROM EMP
ORDER BY SAL DESC;
```



# SQL문법: 조인(JOIN)

# JOIN

## ✓ 정의:

- 두 개 이상의 테이블을 연결하여 데이터를 출력하는 SQL 연산입니다.

## ✓ 키 조건:

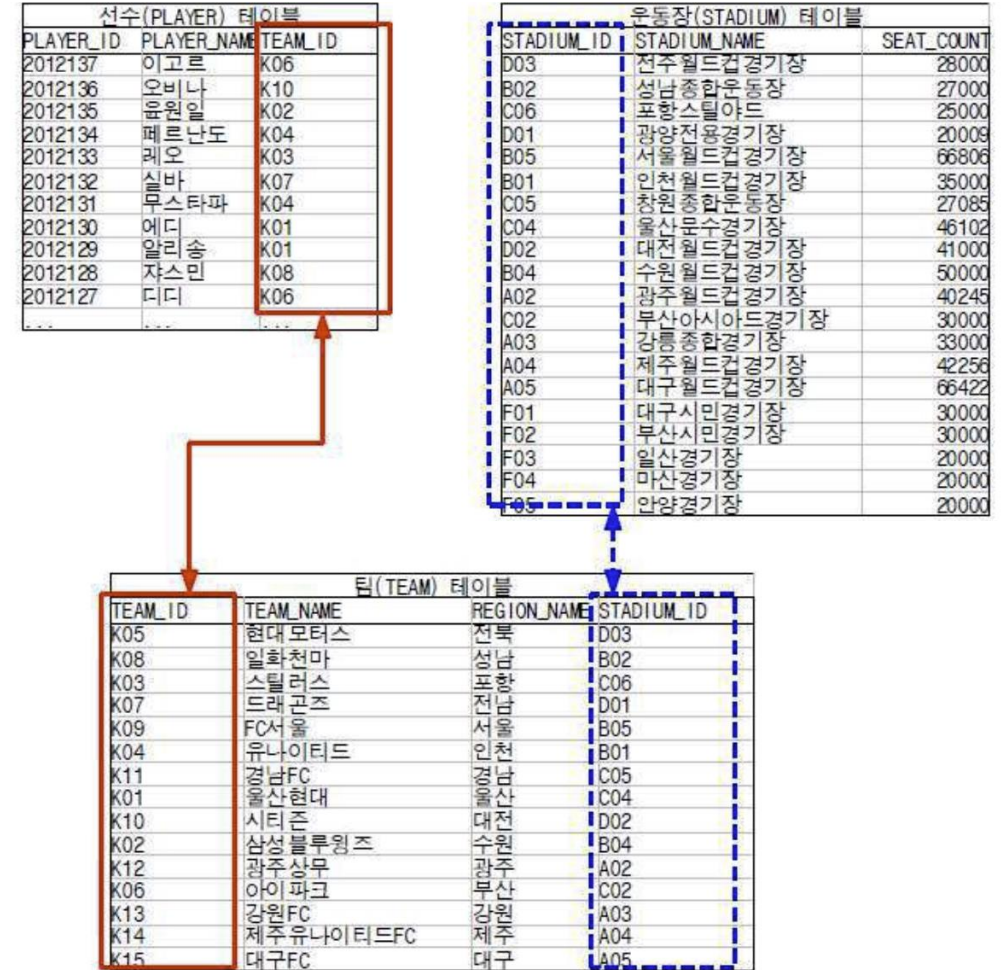
- 주로 PRIMARY KEY(PK) 또는 FOREIGN KEY(FK)를 이용하지만, 이런 관계가 없어도 논리적인 값들의 연관으로 JOIN을 수행할 수 있습니다.

## ✓ 처리 순서:

- 여러 테이블을 FROM 절에 나열하더라도, SQL에서는 두 개의 집합 간에 먼저 조인을 수행하고, 이후 결과 집합과 다른 테이블이 순차적으로 조인됩니다.

## ✓ JOIN 횟수:

- 일반적으로 N개의 테이블을 JOIN 하기 위해서는 최소 (N-1) 번의 JOIN 과정이 필요합니다.



# 등가 / 비등가 / 복수 조인

## ✓ EQUI JOIN (등가 조인)

- 정의: 두 테이블 간 칼럼 값이 서로 정확하게 일치할 때 사용됩니다. 대부분 PK ↔ FK 관계를 기반으로 합니다. ('='으로 조인하는 경우를 의미함)

```
SELECT T1.COL, T2.COL
FROM T1
INNER JOIN T2 ON T1.COL1 = T2.COL2;
```

## ✓ Non EQUI JOIN (비등가 조인)


- 정의: 칼럼 값들이 서로 정확하게 일치하지 않는 경우 사용되며, "=" 대신 다른 연산자들(Between, >, >=, <, <= 등)을 사용합니다.

```
SELECT E.ENAME 사원명, E.SAL 급여, S.GRADE 급여등급
FROM EMP E, SALGRADE S
WHERE E.SAL BETWEEN S.LOSAL AND S.HISAL;
```

## ✓ 3개 이상의 테이블 JOIN

- 정의: 여러 테이블이 연관되어 조인되는 경우, 조인 조건은 필요한 테이블 수에서 하나를 뺀 수만큼 필요합니다. (테이블 간 논리적 연관관계를 통해 JOIN 수행)

```
SELECT P.PLAYER_NAME 선수명, P.POSITION 포지션, T.REGION_NAME 연고지, T.TEAM_NAME 팀명, S.STADIUM_NAME 구장명
FROM PLAYER P, TEAM T, STADIUM S
WHERE P.TEAM_ID = T.TEAM_ID AND T.STADIUM_ID = S.STADIUM_ID
ORDER BY 선수명;
```



테이블 별칭(ALIAS) 사용: 복잡한 SQL에서 가독성을 높이기 위해 테이블명 대신 별칭을 사용할 수 있습니다.

감사합니다