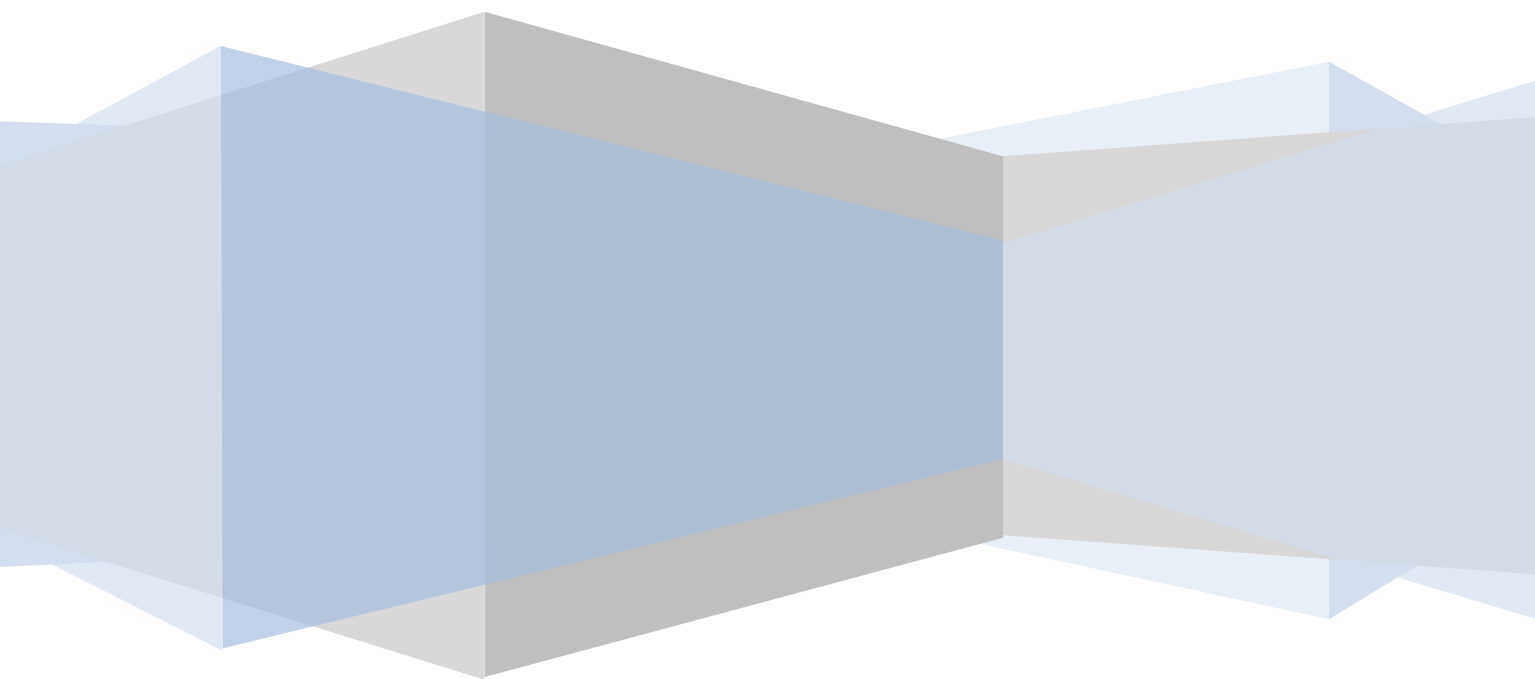


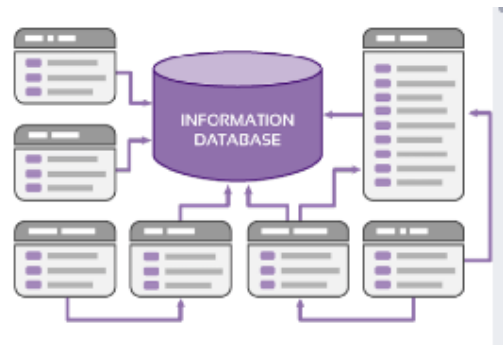
MySQL



데이터베이스 (Database)

● Database 란?

- 다수의 사용자들이 공유해 사용할 수 있도록 통합하고, 저장한 운영 데이터의 집합체
- 데이터를 체계화하여 저장한 공용 데이터들의 묶음
- 쉽게 말해, 구조화된 Data 집합



● Database 의 특징

- 실시간 접근성 : 요구에 따라 실시간으로 처리 응답
- 지속적인 변화 : 삽입, 삭제, 갱신을 통해 데이터의 내용이 계속적으로 변화
- 동시 공유 : 목적이 다른 사용자나 프로그램이 동시에 동일한 데이터를 이용 가능

● Database 의 장점

- 데이터 중복의 최소화
- 데이터의 독립성 유지
- 데이터의 공유
- 데이터의 보안성 유지
- 데이터의 무결성, 일관성 유지

● Database 의 단점

- 운용 비용이 발생
- 오버헤드가 발생

※ overhead 란?

- 어떤 처리를 하기 위해 들어가는 간접적인 처리시간 및 메모리

● Database 용어



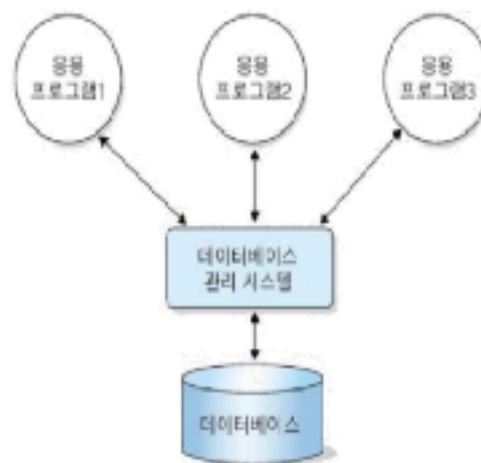
- 식별자 : 관계형 데이터베이스에서 각각의 행(ROW)을 구분할 수 있는 논리적인 개념
 - 유일성 : 하나의 릴레이션에서 모든 행은 서로 다른 키 값을 가져야 한다.
 - 최소성 : 꼭 필요한 최소한의 속성들로만 키를 구성해야 한다.
- 튜플 : 테이블에서의 행을 의미한다. (레코드, ROW)
- 어트리뷰트 : 테이블에서 열을 의미한다. (컬럼)
 - Degree : 어트리뷰트의 수

DBMS (Database Management System)

- 데이터베이스 관리 시스템
- 사용자가 데이터베이스 내의 데이터를 접근할 수 있도록 해주는 소프트웨어 도구

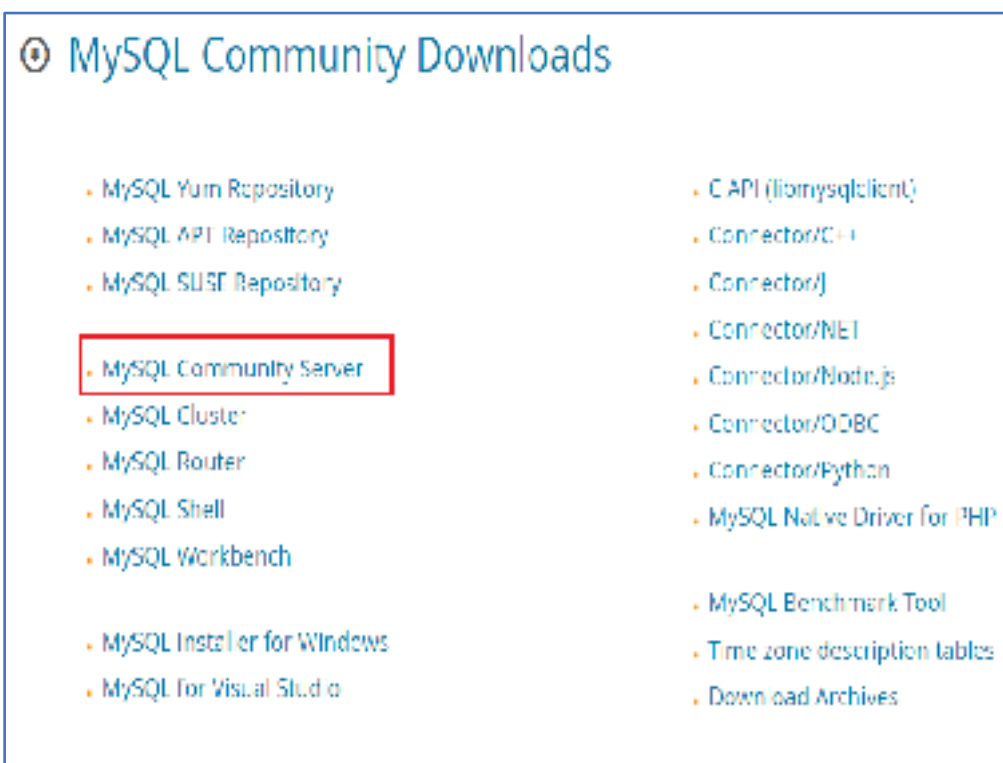
DBMS 의 종류

- Oracle
- MySQL
- MS SQL Server



MySQL 설치 (MySQL Community Server)


1. mysql.org 사이트에 접속한다.
2. 상단 메뉴에서 DOWNLOADS 를 선택한다.
3. 하단에서 MySQL Community(GPL) Downloads >> 항목을 선택한다.
4. MySQL Community Server 를 선택한다.



6. Product Version 을 선택하고 (mysql-installer-community-5.7.19.0.msi)


파일을 다운로드한다.

MySQL Product Archives
MySQL Installer (Archived versions)

 Please note that these are old versions. New releases will have recent bug fixes and features. To download the latest release of MySQL Installer, please visit [MySQL Downloads](#).

Product Version : 5.0.27
Operating System : Microsoft Windows

Windows (x86, 32-bit), MSI Installer mysql-installer-community-5.0.27.msi	Nov 5, 2021	2.3M	Download
Windows (x86, 32-bit), MSI Installer mysql-installer-community-5.0.27.msi	Nov 5, 2021	470.9M	Download

 We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

employees database 설치 (MySQL 제공 학습용)

1. mysql.org 사이트에 접속한다.
2. 상단 메뉴에서 DOCUMENTATION 을 선택한다.

[Percona](#)
[InnoDB Cluster](#)
[MySQL NDB Cluster](#)
[Connectors](#)
[More](#)

Other MySQL Documentation

This page provides additional documentation. There's even more available on these extra pages:

- **Archives:** the documentation archives
- **About:** information about MySQL documentation and the MySQL documentation team

MySQL Server Doxygen Documentation

Title [HTML Online](#)
[MySQL Server \(latest version\)](#) [View](#)

Expert Guides

Language	Title	Version	HTML Online	PDF
English	Extending MySQL	8.0	View	US Ltr A4
English	Extending MySQL	5.7	View	US Ltr A4
English	Extending MySQL	5.6	View	US Ltr A4
English	MySQL Internals		View	
English	MySQL Development Cycle		View	US Ltr A4

Example Databases

Title	DB Download	HTML Setup Guide	PDF Setup Guide
employee data (large dataset, includes data and configuration suite)	GitHub	View	US Ltr A4
world database	TGZ Zip	View	US Ltr A4
world_x database	TGZ Zip	View	US Ltr A4
sakila database	TGZ Zip	View	US Ltr A4

3. GitHub 에서 test_db-master.zip 파일을 다운로드한 후 압축을 푼다.

4. 명령 프롬프트창을 연다.

```
C:\test_db-master> mysql -u root -p -t < employees.sql
```

```
Enter password : ****
```

5. 콘솔에서 로컬 mysql 서버에 접속한다.

```
mysql -u root -p
```

```
Enter password : ****
```

6. mysql 데이터베이스 및 테이블을 조회한다.

```
mysql> show databases;
```

```
-----  
Database  
-----
```

```
employees
```

```
mysql> use employees
```

```
mysql> show tables;
```

데이터베이스 조회하기

```
mysql> show databases ;
```

- 로그인한 계정으로 사용 가능한 데이터베이스 목록을 조회한다.

Database

information_schema

employees

mysql

...

데이터베이스 사용하기

```
mysql> use 데이터베이스명 ;
```

- 사용할 데이터베이스를 선택한다.

테이블 목록 조회하기

```
mysql> show tables;
```

- 현재 선택된 데이터 베이스내의 테이블 목록을 조회한다.

테이블 정보 조회

```
mysql> desc 테이블명 ;
```

- 테이블 정보(컬럼명, 자료형등)를 조회한다.

데이터베이스 생성하기

- 데이터베이스 생성은 CREATE DATABASE 구문을 사용한다.
- 생성한 데이터베이스는 use 문을 사용하여 선택한다.
- utf-8 인코딩 타입의 한글을 저장하려면 다음과 같이 언어 타입을 지정해준다.

```
CREATE DATABASE 데이터베이스 명  
  
CHARACTER SET = 'utf8'  
  
COLLATE = 'utf8_general_ci';
```

현재 사용중인 데이터베이스 조회

```
SELECT DATABASE( );
```

SQL(Structured Query Language)이란?

- SQL 은 관계형 데이터베이스에 대해서 데이터의 구조를 정의, 데이터 조작, 데이터 제어 등을 할 수 있는 절차형 언어이다.
- SQL 은 ANSI/ISO 표준을 준수한다.

SQL 종류

● 데이터 정의를 (DDL : Data Definition Language)

- 관계형 데이터베이스의 구조를 정의하는 언어이다.
- CREATE , ALTER , DROP , RENAME

● 데이터 조작어 (DML : Data Manipulation Language)

- 테이블에서 데이터를 입력, 수정, 삭제, 조회한다.
- INSERT , UPDATE , DELETE , SELECT

● 데이터 제어어 (DCL : Data Control Language)

- 데이터베이스 사용자에게 권한을 부여하거나 회수한다.
- GRANT , REVOKE

● 트랜잭션 제어어 (TCL : Transaction Control Language)

- 트랜잭션을 제어하는 명령문이다.
- COMMIT , ROLLBACK , SAVEPOINT

SQL 을 이용한 데이터 정의

- 테이블을 생성, 변경, 삭제한다.



CREATE TABLE 문 : 테이블 생성

```
CREATE TABLE 테이블_이름 (  
    ❶ 속성_이름 데이터_타입 [NOT NULL] [DEFAULT 기본_값]  
    ❷ [PRIMARY KEY (속성_리스트)]  
    ❸ [UNIQUE (속성_리스트)]  
    ❹ [FOREIGN KEY (속성_리스트) REFERENCES 테이블_이름(속성_리스트)]  
        [ON DELETE 옵션] [ON UPDATE 옵션]  
    ❺ [CONSTRAINT 이름] [CHECK(조건)]  
);
```

- []의 내용은 생략이 가능하다.
- SQL 질의문은 세미콜론(;) 으로 문자의 끝을 표시한다.
- SQL 질의문은 대소문자를 구분하지 않는다.

CREATE TABLE 문 : 테이블 생성

● 속성의 데이터 타입

데이터 타입	의미
char(n)	고정길이 문자열(최대 255byte) n : 자릿수
varchar(n)	가변길이 문자열(65535byte)
int(n)	정수형 타입(4byte) (zerofill)
bigint(n), smallint, tinyint	정수형 타입(8yte, 2byte, 1byte)
float(n,m)	부동소수점 타입(4byte)
double(n,m)	부동소수점 타입(8byte)
date	날짜
time	시간
datetime	날짜와 시간

CREATE TABLE 문 : 테이블 생성

제약 조건 (Constraint)

- 제약 조건이란 데이터의 무결성을 유지하기 위해서 사용된다.
- CREATE 문으로 테이블을 생성하거나 ALTER 문으로 컬럼을 추가할 때 설정한다.
- MySQL 에서 사용할 수 있는 제약 조건은 다음과 같다.

1. NOT NULL
2. UNIQUE
3. PRIMARY KEY
4. FOREIGN KEY
5. DEFAULT

1. NOT NULL

- 속성이 널 값을 허용하지 않는다. 반드시 데이터를 가지고 있어야 한다.
- 예) 고객아이디 VARCHAR(20) NOT NULL

```
CREATE TABLE 테이블명 (  
  
    속성_이름 데이터_타입 NOT NULL ,  
  
    속성_이름 데이터_타입 , -- 널 허용  
  
    ...  
  
)
```

2. DEFAULT

- 속성의 기본 값을 지정하는 키워드이다.
- 예) 적립금 INT DEFAULT 0
- 예) 담당자 VARCHAR(25) DEFAULT '일길동'

```
CREATE TABLE 테이블명 (  
  
    속성_이름 데이터_타입 DEFAULT 기본값 ,  
  
    ...  
  
)
```

3. PRIMARY KEY

- 기본키를 지정하는 키워드이다.
- 해당 속성은 NOT NULL 과 UNIQUE 제약 조건의 특징을 모두 갖는다.
- 예) PRIMARY KEY(고객아이디)

```
CREATE TABLE 테이블명 (  
    속성_이름 데이터_타입 PRIMARY KEY ,  
    ...  
)
```

```
CREATE TABLE 테이블명 (  
    속성_이름 데이터_타입 ,  
    ... ,  
    [CONSTRAINT 제약조건이름] PRIMARY KEY (속성_이름)  
)
```

4. UNIQUE

- 속성은 유일성(중복된 값을 저장할 수 없다)을 가지며, 기본키와 달리 NULL 값이 허용된다.
- 예) UNIQUE(이메일)

```
CREATE TABLE 테이블명 (  
  
    속성_이름 데이터_타입 UNIQUE ,  
  
    ...  
  
)
```

```
CREATE TABLE 테이블명 (  
  
    속성_이름 데이터_타입 ,  
  
    ... ,  
  
    [CONSTRAINT 제약조건이름] UNIQUE (속성_이름)  
  
)
```

5. FOREIGN KEY

- 외래 키라고 부르며, 한 테이블을 다른 테이블과 연결해 주는 역할을 한다.
- 참조 무결성 제약조건을 유지하기 위해서 사용한다.
- 외래키가 어떤 테이블의 무슨 속성을 참조하는지 REFERENCES 키워드 다음에 정의한다.
- 참조되는 테이블의 필드는 반드시 UNIQUE 나 PRIMARY KEY 가 설정되어 있어야 한다.

```
CREATE TABLE 테이블명 (  
    속성_이름 데이터_타입 ,  
    ... ,  
    [CONSTRAINT 제약조건이름]  
    FOREIGN KEY (속성_이름)  
    REFERENCES 테이블명 (속성_이름)  
)
```


5.1. ON DELETE CASCADE

- 자신이 참조하고 있는 테이블의 데이터(ROW)가 삭제되면 자동으로 자신도 삭제되는 옵션이다.
- 참조 무결성 유지하기 위해서 필요한 옵션이다.

5.2. ON UPDATE CASCADE

- 부모 테이블에서 PRIMARY KEY 가 수정될 경우 PRIMARY KEY 를 FOREIGN KEY 로 가지는 값들까지도 같이 변경된다.
- 참조 무결성 유지하기 위해서 필요한 옵션이다.

● 실습

- 고객 테이블은 고객 아이디, 이름, 나이, 등급, 직업, 적립금 속성으로 구성되고, 고객 아이디 속성이 기본키이다. 고객 이름과 등급 속성은 반드시 입력해야 하고, 적립금 속성은 값을 입력하지 않으면 0 이 기본으로 입력되도록 고객 테이블을 생성해 보자.

● 실습

- 제품 테이블은 제품번호, 제품명, 재고량, 단가, 제조업체 속성으로 구성되고, 제품번호 속성이 기본키이다. 재고량 속성은 값을 입력하지 않으면 0이 기본으로 입력되도록 제품 테이블을 생성해 보자.

● 실습

- 주문 테이블은 주문번호, 주문고객, 주문제품, 수량, 배송지, 주문일자 속성으로 구성되고, 주문번호 속성이 기본키이다. 주문 고객 아이디 속성이 고객 테이블의 고객 아이디 속성을 참조하는 외래키이고, 주문 제품 번호 속성이 제품 테이블에서 제품번호 속성을 참조하는 왜라키가 되도록 주문 테이블을 생성해보자.

DROP TABLE 문 : 테이블 삭제

- DROP TABLE 은 테이블의 구조와 데이터를 모두 삭제한다.

DROP TABLE 테이블명 [CASCADE CONSTRAINT]

- CASCADE CONSTRAINT 옵션은 해당 테이블의 데이터를 외래키로 참조한 자식 테이블과 관련된 제약 사항도 삭제할때 사용된다.

ALTER TABLE 문 : 테이블 변경

● 속성 추가 (ADD)

```
ALTER TABLE 테이블_이름  
ADD 속성_이름 데이터_타입 [NOT NULL] [DEFAULT 기본_값];
```

● 실습

- 고객 테이블에 가입날짜 속성을 추가해보자.

~~ALTER TABLE 고객~~

~~ADD 가입날짜 DATETIME NOT NULL DEFAULT NOW();~~

ALTER TABLE 문 : 테이블 변경

● 기존 속성 삭제 (DROP)

```
ALTER TABLE 테이블_이름 DROP 속성_이름 CASCADE | RESTRICT;
```

- CASCADE : 삭제할 속성과 관련된 제약 조건이나 참조하는 다른 속성을 함께 삭제
- RESTRICT : 삭제할 속성과 관련된 제약 조건이나 참조하는 다른 속성이 존재하면 삭제 거부

● 실습

- 연습문제 1 에서 생성한 고객 테이블의 등급 속성을 삭제하면서 관련된 제약 조건이나 등급 속성을 참조하는 다른 속성도 함께 삭제해보자.
- ALTER TABLE 고객 DROP 등급 CASCADE;

SQL 을 이용한 데이터 조작

- 테이블에서 데이터를 검색, 삽입, 수정, 삭제한다.



SELECT 문 : 데이터 검색

● 기본 검색

- SELECT 키워드와 함께 검색하고 싶은 컬럼 이름 나열
- FROM 키워드와 함께 검색하고 싶은 컬럼이 있는 테이블 이름
- 검색 결과는 테이블 형태로 반환됨.

```
SELECT [ ALL | DISTINCT ] 컬럼 리스트  
FROM 테이블명;
```

- ALL : 결과 테이블이 행의 중복을 허용하도록 지정. 생략 가능.
- DISTNCT : 결과 테이블이 행의 중복을 허용하지 않도록 지정.

● 예제

- 고객 테이블에서 모든 속성을 검색해보자.
- 고객 테이블에서 고객 아이디, 고객 이름, 등급 속성을 검색해보자.
- 제품 테이블에서 중복 없이 제조업체 속성을 검색해보자.

SELECT 문 : 데이터 검색

● 기본 검색

- AS 키워드를 이용해 결과 테이블에서 속성의 이름을 바꾸어 출력 가능하다.
- AS 키워드는 생략 가능하다.

● 예제

- 제품 테이블에서 제품명과 단가를 검색하되, 단가를 가격이라는 이름으로 출력해보자.

SELECT 제품명, 단가 AS 가격

FROM 제품 ;

SELECT 문 : 데이터 검색

● 산술식을 이용한 검색

- SELECT 키워드와 함께 산술식으로 (산술 연산자 사용)
- 속성의 값이 실제로 변경되는 것이 아니라 테이블에서 계산된 결과값이 출력된다.

● 예제

- 제품 테이블에서 제품명과 단가를 검색하되, 단가에 500 원을 더해 조정단가라는 새 이름으로 출력해보자.

SELECT 제품명, **단가 + 500** AS 조정단가

FROM 제품 ;

SELECT 문 : 데이터 검색

● 조건 검색

```
SELECT  [ ALL | DISTINCT ] 속성_리스트  
FROM    테이블_리스트  
[ WHERE 조건 ];
```

- WHERE 키워드와 함께 비교 연산자와 논리 연산자를 이용한 검색 조건을 지정한다.
- 숫자 뿐만 아니라 문자, 날짜 값을 비교하는 것도 가능하다.
- 예) 'A' < 'C'
- 예) '2013-12-01' < '2013-12-02'

SELECT 문 : 데이터 검색

● 조건 검색 (비교, 논리 연산자)

비교 연산자	의미
=	같다
<>, !=	다르다
<	작다
>	크다
<=	작거나 같다
>=	크거나 같다

논리 연산자	의미
AND	모든 조건을 만족해야 검색한다.
OR	여러 조건 중 한 가지만 만족해도 검색한다.
NOT	조건을 만족하지 않는 것만 검색한다.

SELECT 문 : 데이터 검색

● LIKE 을 이용한 검색

- 문자열을 이용하는 조건에만 LIKE 키워드 사용 가능하다.

기호	설명
%	0 개 이상의 문자
_	한 개의 문자

사용예	설명
LIKE '데이터%'	데이터로 시작하는 문자열
LIKE '%데이터'	데이터로 끝나는 문자열
LIKE '%데이터%'	데이터가 포함된 문자열
LIKE '데이터____'	데이터로 시작하는 6 자 길이의 문자열
LIKE '___한%'	세번째 글자가 한인 문자열

● 예제

- 고객 테이블에서 성이 김씨인 고객의 이름과 나이, 등급, 적립금을 검색해보자.

SELECT 이름, 나이, 등급, 적립금

FROM 고객

WHERE 이름 LIKE '김%' ;

● 실습

- 고객 테이블에서 고객 아이디가 5 자인 고객의 이름과 나이, 등급, 적립금을 검색해보자.

~~SELECT 이름, 나이, 등급, 적립금~~

~~FROM 고객~~

~~WHERE 고객아이디 LIKE '_____';~~

SELECT 문 : 데이터 검색

● SQL 연산자

BETWEEN A AND B	A 와 B 사이의 값을 조회한다.
IN (list)	OR 을 의미하며 list 값중에 하나만 일치해도 조회된다
IS NULL	NULL 값을 조회한다.

● 부정 SQL 연산자

NOT BETWEEN A AND B	A 와 B 사이의 해당 되지 않는 값을 조회한다.
NOT IN (list)	list 와 불일치한 것을 조회한다.
IS NOT NULL	NULL 이 아닌 값을 조회한다.

SELECT 문 : 데이터 검색

● 정렬 검색 (ORDER BY)

- ORDER BY 키워드와 함께 정렬 기준이 되는 속성과 정렬 방식을 지정한다.
- ASC : 오름차순 (기본) , DESC : 내림차순

```
SELECT [ ALL | DISTINCT ] 속성_리스트  
FROM   테이블_리스트  
[ WHERE 조건 ]  
[ ORDER BY 속성_리스트 [ ASC | DESC ] ];
```

- NULL 값은 오름차순에서 맨 마지막에 출력되고 내림차순에서는 맨 먼저 출력된다.

● 예제

- 고객 테이블에서 이름, 등급, 나이를 검색하되, 나이를 기준으로 내림차순 정렬하시오.

SELECT 이름, 등급, 나이

FROM 고객

ORDER BY 나이 DESC;

● 실습

- 주문 테이블에서 수량이 10 개 이상인 주문의 주문고객, 주문제품, 수량, 주문일자를 검색하시오, 단, 주문제품을 기준으로 오름차순 정렬하고, 동일 제품은 수량을 기준으로 내림차순 정렬해보자.

SELECT 문 : 데이터 검색

- 출력하는 행(ROW)의 개수를 제한하는 LIMIT

```
SELECT [ ALL | DISTINCT ] 컬럼 리스트  
  
FROM 테이블명  
  
[ WHERE 조건 ]  
  
[ ORDER BY 속성 리스트 [ASC | DESC] ]  
  
LIMIT N
```

- 예제

- 고객 테이블에서 이름, 등급, 나이를 검색하되, 나이를 기준으로 내림차순 정렬하시오.
단, 상위 5 개만 출력하시오.

SELECT 이름, 등급, 나이

FROM 고객

ORDER BY 나이 DESC

LIMIT 5;

테이블을 복사하는 CREATE TABLE ... SELECT

CREATE TABLE 새로운테이블 (SELECT 복사할열 FROM 기존테이블)

● 예제

- CREATE TABLE emp (SELECT * FROM employees);
- CREATE TABLE emp (SELECT employee_id, email FROM employees);

MySQL 내장 함수

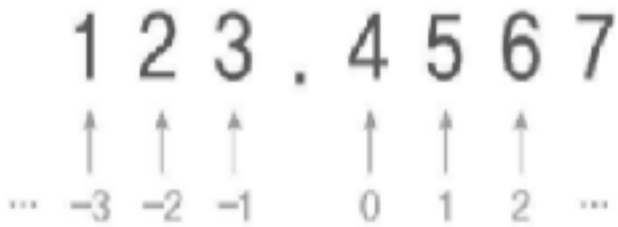
- 숫자 함수
- 문자열 함수
- 날짜 함수
- 제어 흐름 함수
- 형 변환 함수

숫자 함수

ABS(숫자)	절대값을 구한다.
CEIL(숫자)	값보다 큰 정수 중 가장 작은 정수. 소수점 이하 올림
FLOOR(숫자)	값보다 큰 정수 중 가장 큰 정수. 소수점 이하 버림
ROUND(숫자, 자릿수)	자릿수를 기준으로 반올림한다.
TRUNCATE(숫자, 자릿수)	자릿수를 기준으로 버림한다.
POW(x, y)	x 의 y 승
MOD(분자, 분모)	분자를 분모로 나눈 나머지를 구한다.
GREATEST(숫자 1, 숫자 2, ...)	주어진 숫자 중에 가장 큰 값을 반환한다.
LEAST(숫자 1, 숫자 2, ...)	주어진 숫자 중에 가장 작은 값을 반환한다.

- SELECT ABS(-123) ;

● 예제



기분

- SELECT ROUND(123.456, 1) ;
- SELECT ROUND(123.456, -1) ;
- SELECT TRUNCATE(123.456, 1) ;

문자열 함수

1. CONCAT (문자열 1, 문자열 2, 문자열 3, ...)

- 문자열을 합친다.
- SELECT CONCAT('A', 'B', 'C')

2. INSERT (문자열, 시작위치, 길이, 새로운 문자열)

- 문자열의 시작 위치부터 길이 만큼의 문자열을 제거하고 그 자리에 새로운 문자열을 삽입한다.
- SELECT INSERT('AAAAA', 2, 2, 'BBB')
- SELECT INSERT('AAAAA', 2, 0, 'BBB')

3. REPLACE (문자열, 기존 문자열, 새로운 문자열)

- 문자열에서 기존 문자열을 찾아 제거하고 그 자리에 새로운 문자열을 삽입한다.
- SELECT REPLACE('AABBCC', 'BB', 'FF')

4. INSTR (문자열 1, 문자열 2)

- 문자열 1 에서 문자열 2 를 찾아 위치를 반환한다. 위치는 1 부터 시작하며 문자열 2 를 찾지 못하면 0 을 반환한다.
- `SELECT INSTR('ABCDEF', 'CD') ;`

5. LEFT(문자열, 개수)

- 문자열의 좌측부터 개수만큼 가져온다.

6. RIGHT(문자열, 개수)

- 문자열의 우측부터 개수만큼 가져온다.

7. MID(문자열, 시작위치, 개수)

- 문자열에서 시작위치에서 개수만큼 가져온다.

8. SUBSTRING(문자열, 시작위치, 개수)

- 문자열에서 시작위치에서 개수만큼 가져온다.

9. LTRIM(문자열)

- 문자열의 좌측 공백을 제거한다.

10. RTRIM(문자열)

- 문자열의 우측 공백을 제거한다.

11. TRIM(문자열)

- 문자열의 좌우측 공백을 제거한다.

11. LCASE문자열), LOWER(문자열)

- 문자열을 모두 소문자로 변경한다.

12. UCASE문자열), UPPER(문자열)

- 문자열을 모두 대문자로 변경한다.

13. REVERSE(문자열)

- 문자열을 반대로 가져온다.

14. UPPER(문자열), LOWER(문자열)

- 소문자를 대문자, 대문자를 소문자로 변환한다.

15. LPAD(문자열, 길이, 채울 문자열)

RPAD(문자열, 길이, 채울 문자열)

- 문자열을 길이만큼 늘린 후 빈곳을 채울 문자열로 채운다.
- SELECT LPAD('이것이', 5, '##') ;

● 예제

- 사원의 성과 이름을 하나의 문자열로 조회하시오.

```
SELECT CONCAT(last_name, ' ', first_name) AS 'full_name'
```

```
FROM employees ;
```

날짜 함수

1. NOW() , SYSDATE() , CURRENT_TIMESTAMP()

- 현재 날짜와 시간을 반환한다.
- SELECT NOW()

2. CURRENT_DATE() , CURDATE()

- 현재 날짜를 반환한다.
- SELECT CURDATE()

3. CURRENT_TIME() , CURTIME()

- 현재 시간을 반환한다.
- SELECT CURTIME()

4. DATE_ADD (날짜, INTERVAL 기준값)

- 기준값 : (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND)
- 날짜에서 기준값 만큼 더한다.
- SELECT DATE_ADD (NOW(), INTERVAL 100 DAY);

현재 날짜를 기준으로 100일 이후의 날짜를 구한다 .

● 예제

- 전체 사원들이 입사일로부터 100일 이후의 날짜를 조회하시오.

5. DATE_SUB (날짜, INTERVAL 기준값)

- 기준값 : (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND)
- 날짜에서 기준값 만큼 뺀다.

6. YEAR(날짜)

- 날짜에서 연도를 구한다.
- SELECT NOW(), YEAR(NOW()) ;

7. MONTH(날짜)

- 날짜에서 월을 구한다.
- SELECT NOW(), MONTH(NOW()) ;

8. MONTHNAME(날짜)

- 날짜에서 월을 영어로 구한다.
- SELECT NOW(), MONTHNAME(NOW()) ;

9. DAYNAME(날짜)

- 날짜에서 요일을 영어로 구한다.
- SELECT NOW(), DAYNAME(NOW()) ;

10. DAYOFMONTH(날짜)

- 날짜에서 월별 일자를 구한다.
- SELECT NOW(), DAYOFMONTH(NOW()) ;

11. DAYOFWEEK(날짜)

- 날짜에서 요일을 구한다.
- 일요일 - 1 , 월요일 - 2 , 화요일 - 3, ...

12. WEEKDAY(날짜)

- 날짜에서 요일을 구한다.
- 월요일 - 0 , 화요일 - 1 , 수요일 - 2 , ...

13. DAYOFYEAR(날짜)

- 일년을 기준으로 한 날짜까지의 날 수

14. WEEK(날짜)

- 일년 중 몇번째 주

15. FROM_DAYS(날수)

- 00년 00월 00일 부터 날수 만큼 지난 날짜

16. TO_DAYS(날짜)

- 00년 00월 00일 부터 날짜까지의 일 수

17. DATE_FORMAT (날짜, 형식)

- SELECT now(), date_format(now(), '%Y년 %m월 %d일 %H시 %i분 %S초')

%Y	4 자리 년도
%y	2 자리 년도
%m	숫자 월(두 자리)
%d	일자(두 자리)
%H	시간(24 시간)
%i	분
%S	초
%r	hh:mm:ss AM, PM

※ MySQL 날짜 차이 구하기

- MySQL에서 두 날짜간의 차이를 구할때 사용하는 함수가 두 가지가 있다.
- 단순히 일 차이를 구할 때는 DATEDIFF 함수이고, 이 외에도 차이를 연, 분기, 월, 주, 일, 시, 분, 초를 지정하여 가져올 때 사용하는 함수가 TIMESTAMPDIFF 함수이다.
- MySQL에서 두 날짜간의 차이를 구할때 사용하는 함수가 두 가지가 있다.

DATEDIFF (날짜 1, 날짜 2)

- 날짜 1 - 날짜 2 : 두 날짜 간의 차이를 일로 반환한다.
- SELECT DATEDIFF(NOW(), '2022-01-01')

TIMESTAMPDIFF (단위, 날짜 1, 날짜 2)

- 단위 : SECOND(초) , MINUTE(분) , HOUR(시간) , DAY(일) , WEEK(주) ,
MONTH(월) , QUARTER(분기) , YEAR(연도)
- SELECT TIMESTAMPDIFF(DAY, '2022-01-01', NOW())
두 날짜 간의 차이를 '일'로 반환한다.

제어 흐름 함수

● IF (수식, 참, 거짓)

- 수식이 참일 경우 두번째 인수 반환, 거짓일 경우 세번째 인수를 반환한다.

```
SELECT IF (1 > 2, '참', '거짓');
```

● IFNULL (수식 1, 수식 2)

- 수식1이 NULL 이 아니면 수식1 반환, NULL 이면 수식2 반환한다.

```
SELECT IFNULL ( NULL, '널' );
```

```
SELECT IFNULL ( 100, '널' );
```

● NULLIF (수식 1, 수식 2)

- 수식1과 수식2가 같으면 NULL 반환, 다르면 수식1 반환한다.

```
SELECT NULLIF ( 100, 100 ) ;
```

```
SELECT NULLIF ( 100, 200 ) ;
```

● CASE__WHEN__ELSE__END

- 다중 조건문

```
SELECT
```

```
    CASE
```

```
        WHEN 조건 1 THEN 반환값 1
```

```
        WHEN 조건 2 THEN 반환값 2
```

```
        ...
```

```
        ELSE 반환값
```

```
    END AS 별칭
```

```
FROM 테이블명 ;
```

● 예시

```
SELECT employee_id , salary ,  
       CASE  
         WHEN salary >= 20000 THEN '상'  
         WHEN salary >= 15000 THEN '중'  
         WHEN salary >= 10000 THEN '중하'  
         ELSE '하'  
       END AS '급여 수준'  
FROM employees  
LIMIT 10;
```

형변환 함수

● CAST (expr AS type)

- 지정한 값을 다른 데이터 타입으로 변환한다.

- SELECT CAST(20220101 AS DATE) ;

숫자를 날짜 타입으로 변환

- SELECT CAST(20220101 AS CHAR) ;

숫자를 문자열 타입으로 변환

● CONVERT (expr, type)

- 지정한 값을 다른 데이터 타입으로 변환한다.

- SELECT CONVERT(20220101 , DATE) ;

숫자를 날짜 타입으로 변환

- SELECT CONVERT(20220101 , CHAR) ;

숫자를 문자열 타입으로 변환

SELECT 문 : 데이터 검색

● 집계 함수(aggregate function)를 이용한 검색

- 특정 속성 값을 통계적으로 계산한 결과를 검색하기 위해서 집계함수를 이용한다.
- 집계함수는 **NULL 값을 제외하고 계산한다.**
- 집계함수는 WHERE 절이 아닌 SELECT 절에서만 사용 가능하다.

· 집계 함수의 종류

함수	설명	사용가능한 속성 타입
COUNT	속성 값의 개수	모든 데이터
MAX	속성 값의 최대값	
MIN	속성 값의 최소값	
SUM	속성 값의 합계	숫자 데이터
AVG	속성 값의 평균	

● 예제

1. 제품 테이블에서 모든 제품의 단가 평균을 검색하시오.

- SELECT AVG(단가)

FROM 제품 ;

2. 제품 테이블에서 한빛제과에서 제조한 제품의 재고량 합계를 검색해보자.

- SELECT SUM(단가)

FROM 제품

WHERE 제조업체 = '한빛제과' ;

3. 제품 테이블에서 유니크한 제조업체의 수를 검색하시오.

- SELECT COUNT(DISTINCT 제조업체)

FROM 제품 ;

SELECT 문 : 데이터 검색

● 그룹별 검색 (GROUP BY)

```
SELECT [ ALL | DISTINCT ] 속성_리스트  
FROM 테이블_리스트  
[ WHERE 조건 ]  
[ GROUP BY 속성_리스트 [ HAVING 조건 ] ]  
[ ORDER BY 속성_리스트 [ ASC | DESC ] ];
```

- GROUP BY 키워드를 이용해 특정 속성의 값이 같은 튜플(행)을 모아 그룹을 만들고, 그룹별로 검색한다.
- GROUP BY 키워드와 함께 그룹을 나누는 기준이 되는 속성을 지정한다.
- 그룹을 나누는 기준이 되는 속성을 SELECT 절에도 작성하는 것이 좋다.

● 예시

1. 주문 테이블에서 주문 제품별 수량의 합계를 검색하시오.

```
SELECT 주문제품, SUM(수량) AS '총주문수량'
```

```
FROM 주문
```

```
GROUP BY 주문제품 ;
```

2. 제품 테이블에서 제조업체별로 제조한 제품의 개수와 제품 중 가장 비싼 단가를 검색하되, 제품의 개수는 제품수라는 이름으로 출력하고 가장 비싼 단가는 최고가 라는 이름으로 출력하시오.

```
SELECT COUNT(제품번호) AS '제품수' , MAX(단가) AS '최고가' FROM 제품
```

```
GROUP BY 제조업체 ;
```

SELECT 문 : 데이터 검색

● 그룹별 검색 (GROUP BY) + HAVING

● 예제

- 제품 테이블에서 제품을 3 개 이상 제조한 제조업체별로 제품의 개수와 제품 중 가장 비싼 단가를 검색하시오.

```
SELECT 제조업체, COUNT(*) AS 제품수, MAX(단가) AS 최고가  
FROM 제품  
GROUP BY 제조업체  
HAVING COUNT(*) >= 3 ;
```

※ 집계 함수를 이용한 조건은 WHERE 절에서는 작성할 수 없고, HAVING 절에서만 작성 가능하다.

SELECT 문 실행 순서

```
SELECT [ ALL | DISTINCT ] 속성 리스트 ⑤  
FROM 테이블 명 ①  
[ WHERE 조건 ] ②  
[ GROUP BY 속성 리스트 ] ③  
[ HAVING 조건 ] ④  
[ ORDER BY 속성 리스트 [ ASC | DESC ] ]; ⑥
```

1. FROM : 조회 테이블 확인
2. WHERE : 데이터 추출 조건 확인
3. GROUP BY : 그룹화 조건 확인
4. SELECT : 데이터 추출 (어떤 속성을 출력할 지 선택)
5. ORDER BY : 데이터(행) 순서 정렬

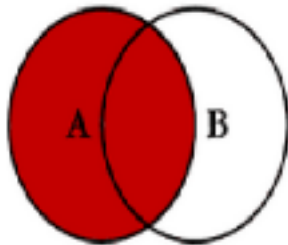
조인 (JOIN)

- 관계형 데이터베이스에서는 중복 데이터를 피하기 위해서 데이터를 쪼개 여러 테이블로 나눠서 저장한다. 이렇게 분리되어 저장된 데이터에서 원하는 결과를 다시 도출하기 위해서는 여러 테이블을 연결하여 데이터를 검색할 필요가 있다. 이런 경우 조인을 사용한다.
- 조인 검색 : 여러 개의 테이블을 연결하여 데이터를 검색하는 것
- 조인 속성 : 조인 검색을 위해 테이블을 연결해주는 속성
 - 연결하려는 테이블 간에 조인 속성의 이름은 달라도 되지만 도메인은 같아야 한다.
 - 일반적으로 외래키(FOREIGN KEY)가 조인 속성으로 이용된다.
- FROM 절에 검색에 필요한 모든 테이블을 나열한다.
- WHERE 절에 조인 속성의 값이 같아야 함을 의미하는 조인 조건을 제시한다.
- 같은 이름의 속성이 서로 다른 테이블에 존재할 수 있기 때문에 속성 이름 앞에 해당 속성이 테이블의 이름을 표시한다.

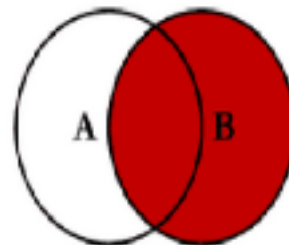
예] 주문.주문고객

SQL JOIN

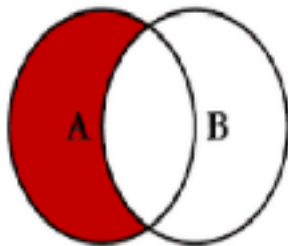
SQL JOINS



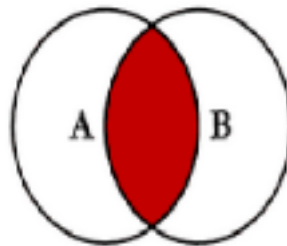
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



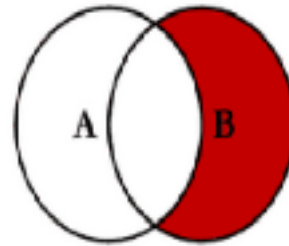
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



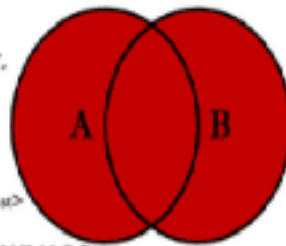
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



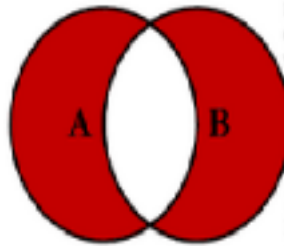
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

© C.L. McFarlin, 2003

※ MySQL 은 FULL OUTER JOIN 을 지원하지 않는다.

● 여러 테이블에 대한 조인 검색



두 개의 테이블을 이용한 조인검색 예 : 주문과 제품 테이블

● 예제

banana 고객이 주문한 제품의 이름을 검색하시오.

SELECT P.제품명

FROM 주문 O , 제품 P

WHERE O.주문고객 = 'banana' AND

O.주문제품 = P.제품번호 ;

주문

주문번호	주문고객	주문제품	수량	배송지	주문일자
o01	apple	c03	10	서울시 대포구	2013-01-01
o02	melon	p01	5	인천시 계곡구	2013-01-10
o03	banana	c08	45	경기도 수원시	2013-01-11
o04	cantal	c02	5	부산시 금정구	2013-02-01
o05	melon	c06	36	경기도 용인시	2013-02-20
o06	banana	p01	19	충청북도 보은군	2013-03-02
o07	apple	c03	22	서울시 영등포구	2013-03-15
o08	pear	c02	50	강원도 춘천시	2013-04-10
o09	banana	c04	15	전라남도 목포시	2013-04-11
o10	cantal	c03	20	경기도 고양시	2013-05-22

고객

고객아이디	고객이름	나이	등급	직업	잔액금
apple	정소화	20	gold	학생	1000
banana	김선우	25	vip	간호사	2500
cantal	고영석	28	gold	교사	4500
orange	김동욱	22	silver	학생	0
melon	성원용	35	gold	회사원	5000
peach	오형준	NULL	silver	의사	300
pear	채광주	31	silver	회사원	500

조건 설정

● 실습

1. 나이가 30 세 이상의 고객이 주문한 제품의 주문제품과 주문일자를 검색하시오.

SELECT O.주문제품 , O.주문일자

FROM 주문 O , 고객 C

WHERE O.주문고객 = C.고객아이디 AND

C.나이 >= 30 ;

2. 고명석 고객이 주문한 제품의 제품명을 검색하시오.

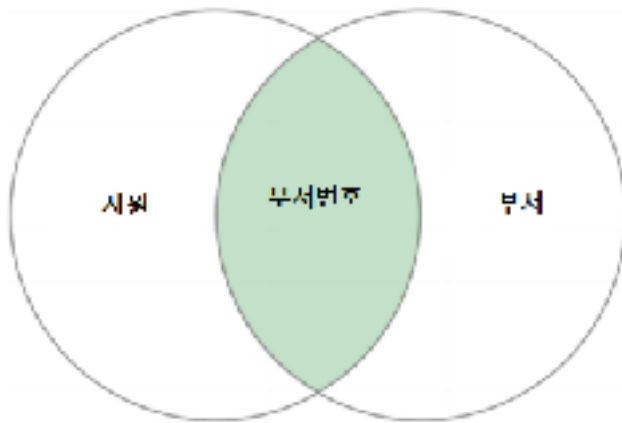
```
SELECT O.주문제품 , O.주문일자  
FROM 주문 O , 고객 C, 제품 P  
WHERE O.주문고객 = C.고객아이디 AND  
O.주문제품 = P.제품번호 AND  
C.고객이름 = '고명석' ;
```

● 다양한 조인 방식 검색

- INNER 조인
- LEFT OUTER, RIGHT OUTER 조인
- SELF 조인
- 카티전 조인 (CROSS 조인)

● EQUI(등가) 조인

- 두 개의 테이블 간에 일치하는 것을 조인한다.
- 조인의 가장 기본은 교집합을 만드는 것이다.



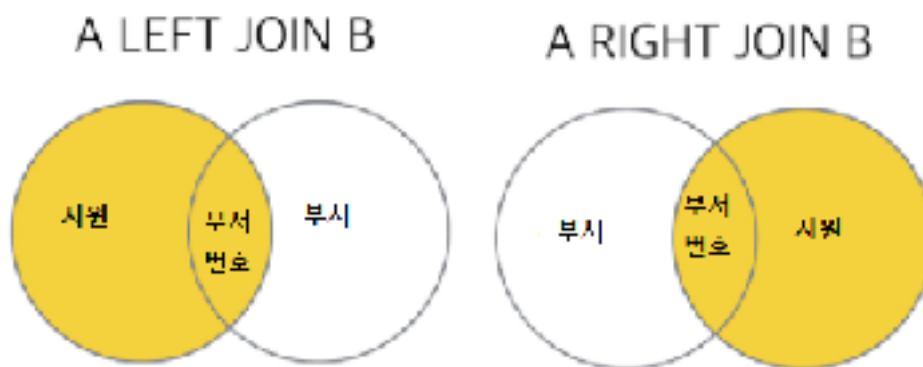
● INNER JOIN (ANSI/ISO 표준)

```
SELECT distinct dept.department_id
FROM employees AS emp
INNER JOIN departments AS dept
ON emp.department_id = dept.department_id ;
```

● LEFT OUTER , RIGHT OUTER 조인

- LEFT OUTER JOIN : 두 개의 테이블에서 같은 것을 조회하고, 왼쪽 테이블에만 있는 것을 포함해서 조회한다.
- RIGHT OUTER JOIN : 두 개의 테이블에서 같은 것을 조회하고, 오른쪽 테이블에만 있는 것을 포함해서 조회한다.

```
SELECT emp.employee_id, dept.department_id
FROM employees AS emp
LEFT OUTER JOIN departments AS dept
ON emp.department_id = dept.department_id ;
```



● Multiple 조인

```
SELECT emp.employee_id, dept.department_name, job.job_title  
FROM employees emp  
INNER JOIN departments dept  
ON emp.department_id = dept.department_id  
INNER JOIN JOBS job  
ON emp.job_id = job.job_id ;
```

● SELF 조인

- 같은 테이블 내의 데이터를 조인한다. 즉, 하나의 테이블이 자기 자신을 대상으로 조인하는 것을 말한다.

```
SELECT mgr.employee_id '관리자 사번', emp.employee_id '사번'  
FROM employees mgr  
SELF JOIN employees emp  
ON mgr.employee_id = emp.manager_Id;
```

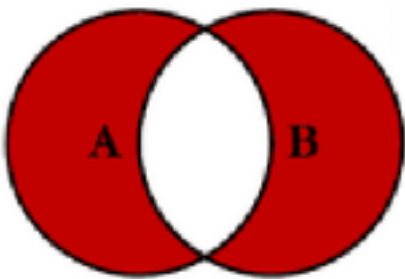
SET (집합)

- 두 SELECT 문을 통해 얻어온 결과를 집합 연산을 통해 하나의 결과로 만드는 것을 set 이라고 부른다.
- 합집합, 교집합, 차집합 등 집합 연산을 할 수 있다.
- 집합 연산을 하기 위해서는 두 SELECT 문을 통해서 가져오는 컬럼이 같아야 한다.

● UNION 을 사용한 합집합 구현

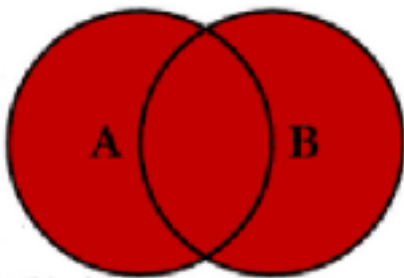
1) UNION

- 두 개의 테이블을 하나로 합치면서 중복된 데이터를 제거하여 하나만 가져온다.
- 두 개의 테이블의 컬럼 수, 컬럼의 데이터 형식이 모두 일치해야 한다.

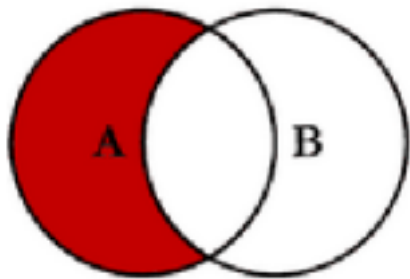


2) UNION ALL

- 두 개의 테이블을 하나로 합치는 것이다. UNION 처럼 중복을 제거하거나 정렬하지 않는다.



● 차집합을 만드는 MINUS



※ MySQL 은 minus 연산자를 지원하지 않는다.

1. LEFT OUTER JOIN 을 이용하는 방법

```
SELECT distinct dept.department_id  
FROM departments dept  
LEFT JOIN employees emp  
ON emp.department_id = dept.department_id  
WHERE emp.department_id IS NULL;
```

2. NOT EXISTS 을 이용하는 방법

```
SELECT dept.department_id
FROM departments dept
WHERE NOT EXISTS (
    SELECT 1
    FROM employees emp
    WHERE emp.department_id = dept.department_id
);
```

3. NOT IN 을 이용하는 방법

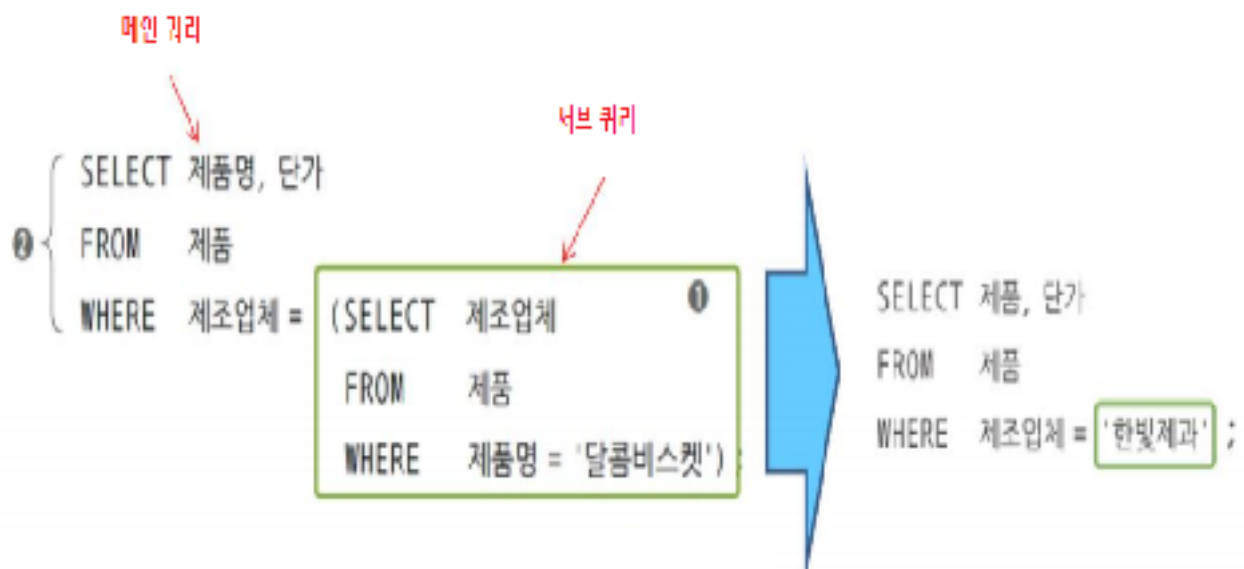
```
SELECT department_id
FROM departments
WHERE department_id NOT IN ( SELECT distinct department_id
                              FROM employees
                              WHERE department_id IS NOT NULL ) ;
```

Sub Query (서브 쿼리)

- SELECT 문 안에 또 다른 SELECT 문을 사용하는 SQL 문이다.
- Main Query : 다른 SELECT 문을 포함하는 SELECT 문
- Sub Query : 다른 SELECT 문 안에 내포된 SELECT 문
- 서브 쿼리는 괄호로 묶어서 작성, ORDER BY 절을 사용할 수 없다.
- 서브 쿼리를 먼저 수행하고, 그 결과를 이용해 메인 쿼리를 수행한다.

● 예시

- 달콤비스켓과 같은 제조업체에서 제조한 제품의 제품명과 단가를 검색해보자.



● 반환 값에 따른 서브 쿼리

- 단일 행 서브쿼리는 비교 연산자(=, >, <, >=, <=) 사용 가능.
- 다중 행 서브쿼리는 비교 연산자 사용이 불가능.

※ 다중 행 서브쿼리에 사용 가능한 연산자

연산자	설명
IN	서브 쿼리 결과 값 중 일치하는 것이 있으면 검색 조건이 참
NOT IN	서브 쿼리 결과 값 중 일치하는 것이 없으면 검색 조건이 참
EXISTS	서브 쿼리 결과 값이 하나라도 존재하면 검색 조건이 참
NOT EXISTS	서브 쿼리 결과 값이 하나도 존재하지 않으면 조건이 참
ALL	서브 쿼리 결과 값 모두와 비교한 결과가 참이면 검색 조건을 만족 (비교 연산자와 함께 사용)
ANY 또는 SOME	서브 쿼리 결과 값 중 하나라도 비교한 결과가 참이면 검색 조건을 만족 (비교 연산자와 함께 사용)

● 예시

- 적립금이 가장 많은 고객의 이름과 적립금을 검색하시오.

①

```
SELECT MAX(적립금) FROM 고객 ; // 5000
```

②

```
SELECT 이름과 적립금
```

```
FROM 고객
```

```
WHERE 적립금 = 5000;
```

- 서브 쿼리를 실행한 경우

```
SELECT 이름과 적립금
```

```
FROM 고객
```

```
WHERE 적립금 = ( SELECT MAX(적립금) FROM 고객 ) ;
```

● 실습

- banana 고객이 주문한 제품의 제품명과 제조업체를 검색하시오.

```
SELECT 제품명, 제조업체
FROM 제품
WHERE 제품번호 IN ( SELECT 주문제품
                     FROM 주문
                     WHERE 주문고객 = 'banana' )
```

- banana 고객이 주문하지 않은 제품의 제품명과 제조업체를 검색하시오.

```
SELECT 제품명, 제조업체
FROM 제품
WHERE 제품번호 NOT IN ( SELECT 주문제품
                        FROM 주문
                        WHERE 주문고객 = 'banana' )
```

● 실습

- 대한식품이 제조한 모든 제품의 단가 보다 비싼 제품의 제품명, 단가, 제조업체를 검색하시오.

```
SELECT 제품명, 단가, 제조업체
FROM 제품
WHERE 단가 > ALL ( SELECT 단가
                   FROM 제품
                   WHERE 제조업체 = '대한식품' ) ;
```

- 2013 년 3 월 15 일에 제품을 주문한 고객의 이름을 검색하시오.

```
SELECT 이름
FROM 고객
WHERE 고객번호 IN ( SELECT 제품고객
                   FROM 주문
                   WHERE 주문일자 = '2013-03-15' ) ;
```

Multiple column Subquery

- 서브 쿼리 결과가 여러 개의 컬럼인 경우

● 실습.

· 부서별 최고 급여를 받는 사원 정보를 조회하시오

```
SELECT
FROM employees
WHERE (department_id, salary) IN ( SELECT department_id, MAX(salary)
                                from employees
                                GROUP BY department_id ) ;
```


Correlative Subquery (연관 서브 쿼리)

- 서브 쿼리 내에서 메인 쿼리 컬럼이 사용된 서브쿼리이다.

● 실습.

· 부서별 평균 급여보다 급여를 많이 받는 사원 정보를 조회하시오

```
SELECT department_id, employee_id, salary
FROM employees e
WHERE salary > ( SELECT AVG(salary)
                  FROM employees
                  WHERE department_id = e.department_id )
ORDER BY department_id ASC, employee_Id ASC ;
```

※ 실행 순서

1. 메인 쿼리가 먼저 실행된다.
2. 후보행이 선택, 후보값이 서브 쿼리 공급
3. 서브 쿼리 실행, 결과를 리턴한다.
4. 리턴 받은 결과를 가지고 조건을 비교한다.
5. 다음 후보행이 없을 때 까지 2~ 4 단계 반복 수행한다.

Scalar Subquery

- SELECT 문에 오는 서브 쿼리를 스칼라 서브쿼리라고 한다.
- 단일 행과 컬럼을 리턴한다.

● 실습.

- 모든 사원 정보를 조회하시오 (사번, 급여, 부서명)

```
SELECT employee_id, salary ,  
       ( SELECT department_name FROM departments d  
         WHERE d.department_id = e.department_id ) AS '부서명'  
FROM employees e  
ORDER BY employee_id ASC ;
```

INLINE VIEW

- FROM 절에 오는 서브 쿼리를 인라인 뷰라고 한다.
- 여러개의 행과 컬럼을 리턴할 수 있다.
- 상호 연관 서브 쿼리 사용이 불가능하다.
- ORDER BY 절에 사용이 가능하다.

● 실습.

- 부서 정보를 조회하시오. (부서별 총 급여)

```
SELECT d.*, e.sum_salary
FROM departments d , ( SELECT department_id, SUM(salary) as sum_salary
                        FROM employees
                        GROUP BY department_id ) e
WHERE d.department_id = e.department_id ;
```

```
SELECT d.*, e.sum_salary
FROM departments d
INNER JOIN ( SELECT department_id, SUM(salary) as sum_salary
             FROM employees
             GROUP BY department_id ) e
ON d.department_id = e.department_id ;
```

- 부서별 평균 급여보다 총 급여가 높은 부서 정보를 조회하시오.

```
SELECT department_id, SUM(salary), AVG(salary)
FROM employees e
GROUP BY department_id
HAVING SUM(salary) > (SELECT AVG(salary)
                     FROM employees
                     GROUP BY department_id
                     HAVING department_id = e.department_id) ;
```

INSERT 문 : 데이터 삽입

● 데이터 직접 삽입

```
INSERT  
INTO 테이블명 [(속성 리스트)]  
VALUES (속성값 리스트) ;
```

- INTO 키워드와 함께 삽입할 테이블의 이름과 속성 이름을 나열한다.
속성 리스트를 생략하면 테이블을 정의할 때 저장한 속성 순서대로 값이 삽입된다.
- VALUES 키워드와 함께 삽입할 속성 값들을 나열한다.
- INTO 절의 속성 이름과 VALUES 절의 속성 값은 순서대로 일대일 대응 되어야 한다.

● 예시

- 고객 테이블에 아이디가 strawberry, 이름이 일길동, 나이가 25 세, 등급이 vip, 직업이 프로그래머, 적립금이 100 원인 새로운 고객 정보를 삽입하시오.

```
INSERT INTO 고객 (아이디, 이름, 나이, 등급, 직업, 적립금)
VALUES ('strawberry', '일길동', 25, 'vip', '프로그래머', 100) ;
```

● 실습

- 고객 테이블에 아이디가 peach, 이름이 아이유, 나이가 30 세, 등급이 vvip, 직업은 아직 모르고 , 적립금이 500 원인 새로운 고객 정보를 삽입하시오.

INSERT 문 : 데이터 삽입

● Sub Query 를 이용한 데이터 삽입

- SELECT 문을 이용해 다른 테이블에서 검색한 데이터를 삽입한다.

```
INSERT  
INTO 테이블명 [(속성 리스트)]  
SELECT 문 ;
```

● 예시

```
INSERT INTO 한빛제품 (제품명, 재고량, 단가)  
SELECT 제품명, 재고량, 단가  
FROM 제품  
WHERE 제조업체 = '한빛제과';
```


UPDATE 문 : 데이터 수정

```
UPDATE 테이블명  
SET 속성이름 1 = 값 1, 속성이름 2 = 값 2, ...  
[ WHERE 조건 ] ;
```

- SET 키워드 다음에 속성 값을 어떻게 수정할 것인지를 지정한다.
- WHERE 절에 제시된 조건을 만족하는 튜플(행)에 대해서만 속성값을 수정한다.
단. WHERE 절이 생략되면 테이블에 존재하는 모든 튜플(행)이 수정된다.

● 예시

- 제품 테이블에서 제품 번호가 p30 인 제품의 제품명을 통큰파이로 수정하시오.

```
UPDATE 제품  
SET 제품명 = '통큰파이'  
WHERE 제품번호 = 'p30' ;
```

● 실습

1. 제품 테이블에 있는 모든 제품의 단가를 10% 인상하시오.

UPDATE 제품

SET 단가 = 단가 * 1.1 ;

2. 정소화 고객이 주문한 제품의 주문수량을 5 개로 수정하시오.

```
UPDATE 주문
```

```
SET 주문수량 = 5
```

```
WHERE 주문고객 IN ( SELECT 고객번호
```

```
FROM 고객
```

```
WHERE 이름 = '정소화' );
```

DELETE 문 : 데이터 삭제

```
DELETE  
FROM 테이블명  
[ WHERE 조건 ] ;
```

- WHERE 절에 제시한 조건을 만족하는 튜플(행)에 대해서만 삭제한다.
단. WHERE 절이 생략되면 테이블에 존재하는 모든 튜플(행)을 삭제한다.

● 예시

- 주문 테이블에서 주문 일자가 2013 년 5 월 22 일인 주문내역을 삭제하시오.

```
DELETE  
FROM 주문  
WHERE 주문일자 = '2013-05-22' ;
```

뷰 (View)

- 데이터베이스에 존재하는 일종의 가상 테이블이다.
- 뷰는 단지 다른 테이블이나 다른 뷰에 있는 데이터를 보여주는 역할만을 수행한다.

● 뷰의 특징

- 특정 사용자에게 테이블 전체가 아닌 필요한 속성만 보여줄 수 있다.
- 복잡한 쿼리를 단순화해서 사용할 수 있다.

● 뷰 생성 및 대체

```
CREATE [ OR REPLACE ] VIEW 뷰이름 AS  
SELECT 속성이름 1, 속성이름 2 [ AS 새로운 속성이름 ], ...  
FROM 테이블명  
WHERE 조건
```

● 뷰 삭제

```
DROP VIEW 뷰이름
```

● 예시

```
CREATE VIEW dept_emp_latest_date AS
SELECT emp_no,
       MAX(from_date) AS 'from_date',
       MAX(to_date) AS 'to_date'
FROM dept_emp
GROUP BY emp_no ;

SELECT * FROM dept_emp_latest_date ;
```

● 예시

```
CREATE VIEW current_dept_emp_view AS  
SELECT d.*  
FROM dept_emp d  
INNER JOIN dept_emp_latest_date l  
ON d.emp_no = l.emp_no  
    AND d.from_date = l.from_date  
    AND d.to_date = l.to_date ;  
  
SELECT * FROM current_dept_emp_view ;
```

인덱스

- 데이터를 좀 더 빠르게 찾을 수 있도록 해주는 도구이다.

● 장점

- 검색 속도가 빨라질 수 있다. (항상 그런것은 아니다.)
- 해당 쿼리의 부하가 줄어들어서 시스템 전체의 성능이 향상된다.

● 단점

- 인덱스가 데이터베이스 공간을 차지한다. (대략 10%정도의 추가 공간이 필요)
- 처음 인덱스 생성시 시간이 많이 소요될 수 있다.
- 데이터 변경 작업이 자주 일어날 경우 오히려 성능이 나빠질 수도 있다.

● 인덱스를 만들 때 어떤 기준으로 만드는데 좋을까?

1. 크기가 큰 테이블만 만든다.

테이블 크기 때문에 성능이 떨어진다고 싶을 때가 인덱스가 필요하다는 신호이다.

2. PRIMARY KEY , UNIQUE 제약조건이 부여된 컬럼에는 자동으로 인덱스가 생성된다. (Clustered Index)

3. Cardinality(카디널리티)가 높은 컬럼에 만든다.

인덱스를 만드는 열을 결정하는 지침으로써 가장 중요한 것이 카디널리티이다.

'Cardinality'란 '값의 분산도'를 뜻한다. 특정 열에 대해 많은 종류의 값을 가지고 있다면 Cardinality가 높다. 하지만, 값의 종류가 적으면 Cardinality가 낮다는 의미이다. 주민등록번호는 Cardinality가 높지만, 성별은 Cardinality가 낮다.

● 효율적인 INDEX

- WHERE 절에 자주 등장하는 컬럼을 인덱스로 설정
- ORDER BY 절에 자주 등장하는 컬럼을 인덱스로
- SELECT 절에 자주 등장하는 컬럼들을 잘 조합해서 인덱스로 구성
- JOIN 이 자주 사용되는 열에 인덱스를 생성해주는 것이 좋다.

~~SELECT 절에 자주 등장하는 컬럼들을 잘 조합해서 INDEX 로 만들어두면 INDEX 조회 후 다시 데이터에서 조회할 필요가 없으므로 빠르게 검색이 가능하다.~~

● INDEX 가 안 되는 쿼리

· 컬럼을 가공

ex) WHERE SUBSTR(ORDER_NO, 1,4) = '2019' ->

WHERE ORDER_NO LIKE '2019%'

· 인덱스 컬럼의 묵시적 형변환(같은 타입으로 비교해야함)

ex) WHERE REG_DATE = '20220101' ->

WHERE REG_DATE = CONVERT('20220101', DATE)

· 인덱스 컬럼 부정형 비교.

ex) WHERE MEM_TYPE != '10' ->

WHERE MEM_TYPE IN('20', '30')

· %가 앞에 위치.

or 조건 사용 -> UNION ALL 로 대체

※ INDEX 손익분기점

테이블이 가지고 있는 전체 데이터양의 10% ~ 15%이내의 데이터가 출력 될 때만 INDEX를 타는게 효율적이고, 그 이상이 될 때에는 오히려 풀스캔이 더 빠르다.

● 인덱스 종류

- MySQL 은 Clustered INDEX 와 Secondary Index 를 지원한다.
- 클러스터형 인덱스는 테이블당 한 개만 생성할 수 있고, 보조 인덱스는 테이블당 여러개를 생성할 수 있다.

1. Clustered Index

- 인덱스 생성시에 데이터 페이지 전체가 다시 정렬되므로 서비스 운영중에 클러스터형 인덱스를 생성하면 큰 부하가 발생한다.
- 인덱스 자체의 리프 페이지가 곧 데이터이므로 인덱스에 데이터가 포함되어 있다.
- 보조 인덱스보다 검색속도가 빠르나 데이터 변경이 많은 경우는 느리다.
- 테이블에 한 개만 생성할 수 있다.

2. Seondary Index

- 인덱스 생성시 데이터페이지는 건드리지 않고 별도의 페이지에서 인덱스를 구성하는 작업을 실행한다.
- 보조 인덱스의 인덱스 자체의 리프 페이지는 데이터 페이지의 주소값을 갖는다.
- 클러스터형 인덱스보다 검색 속도는 느리고, 데이터 변경 시 덜 느리다.
- 보조 인덱스는 여러 개 생성할 수 있지만, 충분히 고려하고 사용해야 한다.

※ 제약 조건으로 자동 생성되는 인덱스 조회

```
SHOW INDEX FROM 테이블명 ;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment
employees	0	PRIMARY	1	employee_id	A	107	NULL	NULL		BTREE	
employees	0	emp_email_uk	1	email	A	107	NULL	NULL		BTREE	
employees	0	emp_emp_id_pk	1	employee_id	A	107	NULL	NULL		BTREE	
employees	1	emp_dept_fk	1	department_id	A	12	NULL	NULL	YES	BTREE	
employees	1	emp_job_fk	1	job_id	A	19	NULL	NULL		BTREE	
employees	1	emp_manager_fk	1	manager_id	A	19	NULL	NULL	YES	BTREE	

- Non_unique 열은 0 이면 Unique 인덱스를 1 이면 Nonunique 인덱스를 의미한다.
- Key_name 은 인덱스 이름이다.
- Key_name 이 PRIMARY 로 표기되면 Clusterd Index 이고, 나머지는 Secondary Index 이다.
- Null 은 널값의 허용여부이다.
- Cardinality 는 중복되지 않은 데이터 개수가 들어 있다.
- Index_type : MySQL 은 기본적으로 B-Tree 인덱스 구조를 갖는다.

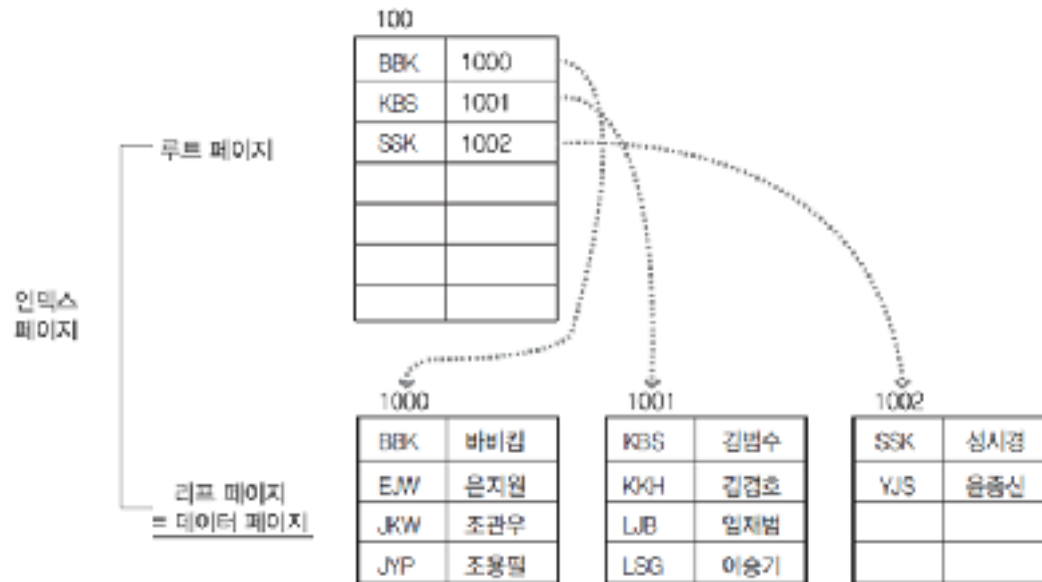
테이블 구조

1. 인덱스가 없을 때

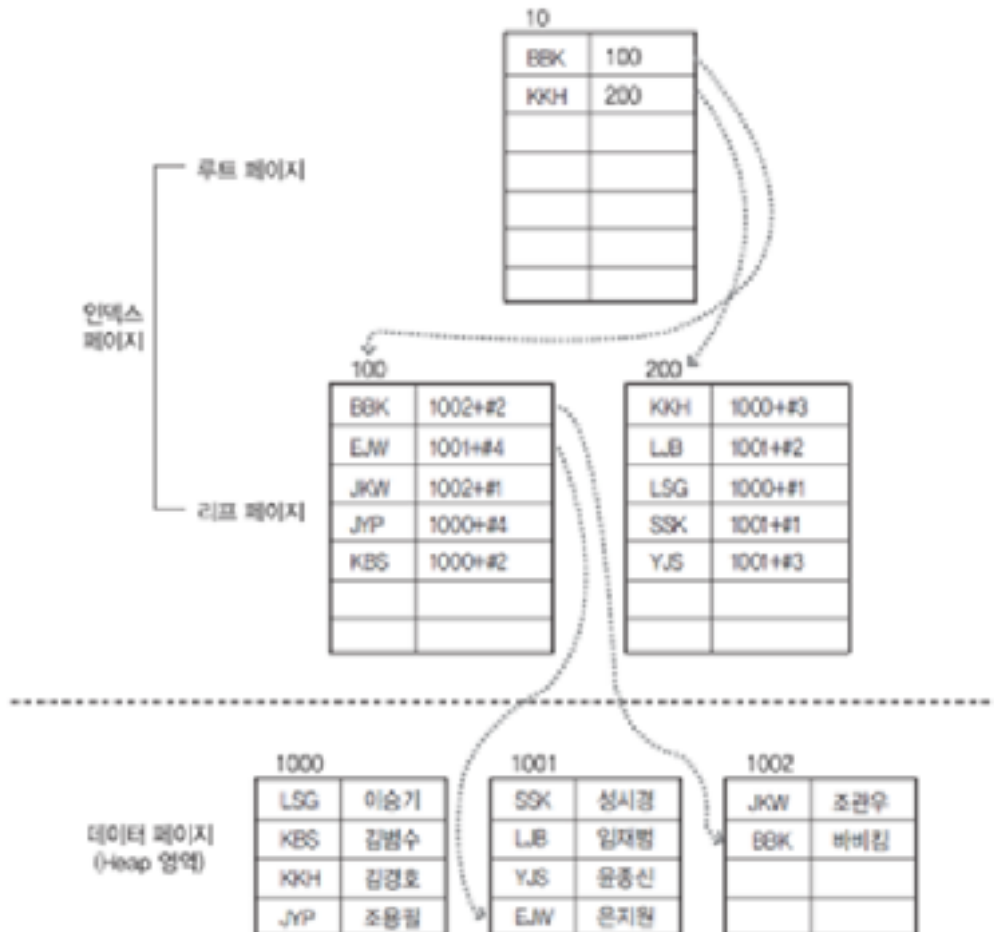
- 인덱스가 없으면 데이터 페이지에 정렬 없이 순서대로 데이터가 저장된다.

데이터 페이지 (Heap 영역)	1000	LSG 이승기	1001	SSK 성시경	1002	JKW 조관우
		KBS 김범수		LJB 임재범		BBK 바비킴
		KKH 김경호		YJS 윤종신		
		JYP 조용필		EJW 은지원		

2. Clustered Index 만 있을 때



3. Secondary Index 만 있을 때



4. Clustered Index, Secondary Index 둘 다 있을 때

- 컬럼의 자리수가 작은 것을 클러스터형 인덱스에 저장하는 것이 바람직하다.
- 클러스터형 인덱스를 지정한 컬럼의 내용은 보조 인덱스에도 저장되기 때문에 차지하는 공간이 커지기 때문이다.

● 인덱스 생성

```
CREATE [UNIQUE] INDEX 인덱스명 ON 테이블명 (컬럼명 1, 컬럼명 2, ...);
```

● 인덱스 보기

```
SHOW INDEX FROM 테이블명 ;
```

● 인덱스 삭제

```
ALTER TABLE 테이블명 DROP INDEX 인덱스명 ;
```

```
ANALYZE TABLE 테이블명 ; //인덱스 생성시 즉시 테이블에 반영  
SHOW TABLE STATUS LIKE '테이블명' ;
```

사용자 계정 생성 / 권한 부여

● 사용자 계정 확인하기

```
USE mysql ; --중요 (데이터베이스명 : mysql)

SELECT host, user FROM user;
```

● 사용자 계정 생성하기

```
CREATE USER 계정 ID@localhost identified by '비밀번호' ;
```

- localhost 만 추가된 계정에 외부 host 접근 권한 추가

```
CREATE USER 계정 ID@ '%' identified by '비밀번호' ;
```

● 계정 권한 부여

- 모든 DB, 테이블 관리 권한 부여
- `privileges on [데이터베이스 이름].[테이블 이름]`

```
grant all privileges on *.* to 계정 ID@localhost  
identified by '비밀번호' ;
```

```
mysql>grant all privileges on *.* to java@'%' identified by 'java1234' ;
```

● 권한 부여 내용 메모리에 반영하기

```
flush privileges ;
```

● 권한 부여 내용 메모리에 반영하기

- show grants for '계정명'@'호스트'

```
show grants for 계정 ID@localhost ;
```

● 계정 삭제

```
drop user 계정 ID@localhost ;
```

● 권한 제거

```
revoke all on '스키마명'.'테이블명' from '계정명'@'호스트' ;
```

트랜잭션

- 데이터베이스에서 데이터 처리의 한 단위를 트랜잭션이라고 한다.
- 대부분의 데이터베이스는 데이터를 저장하고, 삭제하고, 수정하는 작업을 바로 물리적인 하드 디스크에 저장된 데이터에 반영하지 않는다.
- 트랜잭션 기능은 MySQL의 엔진에 영향을 받는다. DB 엔진을 InnoDB로 하셔야 트랜잭션을 사용할 수 있다.

● 트랜잭션 설정

```
mysql> show variables like '%commit%' ;
```

Variable_name	Value
autocommit	ON
binlog_group_commit_sync_delay	0
binlog_group_commit_sync_no...	0

```
-- autocommit off
```

```
mysql> SET AUTOCOMMIT = 0;
```

```
-- autocommit on
```

```
mysql> SET AUTOCOMMIT = 1;
```

- DDL(CREATE, DROP, ALTER, RENAME, TRUNCATE)문은 트랜잭션의 롤백 대상이 아니다.

```
START TRANSACTION or BEGIN ;    -- 트랜잭션 시작

DML 문 ( INSERT, UPDATE, DELETE )
...

COMMIT / ROLLBACK ;            -- 트랜잭션 종료
```

- COMMIT ; 현재 트랜잭션 커밋 (변경 사항 영구 적용)
- ROLLBACK; 현재 트랜잭션 롤백 (변경 사항 초기화)

- SAVEPOINT 를 설정해서 SAVAPPOINT 이전 까지의 작업들만 롤백할 수 있다.

```
SAVEPOINT identifier

ROLLBACK TO identifier
```

스토어드 프로시저 (Stored Procedure)

- 일련의 SQL 문장을 선언해서 MySQL 에 저장하고, 해당 SQL 문을 함수처럼 사용하는것으로 만들어 두기만 하면 함수처럼 호출하여 편하게 사용할 수 있다.

● 스토어드 프로시저 생성

```
DELIMITER $$  
CREATE PROCEDURE 프로시저명 (  
    [IN, OUT, INOUT] 변수명 자료형 , ...  
)  
BEGIN  
  
    SQL 문  
  
END $$  
DELIMITER ;
```

- IN : 스토어드 프로시저에 값을 전달한다.
- OUT : 스토어드 프로시저의 값을 호출자에게 반환한다.
- INOUT : IN + OUT

- 스토어드 프로시저 내부에 사용하는 SQL 문은 일반 SQL 문이기때문에 세미콜론(;)으로 문장을 끝맺어야 한다. 이 때, 저장 프로시저 작성이 완료되지 않았음에도 SQL 문이 실행되는 위험을 막기 위해 구분자(;)를 다른 구분자로 바꿔주어야하는데 이 때 사용하는 명령어가 DELIMITER 이다.

따라서 스토어드 프로시저 생성 전에 구분자(DELIMITER)를 \$\$ 으로 바꾸주고 프로시저 작성이 끝났을 때 END \$\$ 로 저장 프로시저의 끝을 알려준다. 마지막으로 구분자를 원래대로 되돌리기 위해 구분자(DELIMITER)를 세미콜론(;)으로 바꿔준다.

● 스토어드 프로시저 호출

```
CALL 스토어드 프로시저명 ;
```

```
DECLARE @NUM = 0 ;           -- 변수 초기화
SET @NUM = 10;               -- 변수에 값 할당
CALL test_proc (1, 2, @NUM); -- 스토어드 프로시저 호출
SELECT @NUM;                 -- 변수 값 출력
```

● 스토어드 프로시저 목록 확인

```
SHOW PROCEDURE STATUS WHERE DB = '데이터베이스명' ;
```

● 스토어드 프로시저 내용 확인

```
SHOW CREATE PROCEDURE 프로시저 이름 ;
```

● 스토어드 프로시저 삭제

```
DROP PROCEDURE 프로시저 이름 ;
```

● 예시

- 고객 아이디에 해당하는 고객 이름과 나이를 조회하는 스토어드 프로시저를 생성해보자. (retrieveCustomerById)

```
DELIMITER $$

CREATE PROCEDURE retrieveCustomerById (

    IN      v_id      VARCHAR(50),

    OUT     v_name     VARCHAR(25),

    OUT     v_age      INT

)

BEGIN

    SELECT name , age

    INTO v_name , v_age

    FROM customer

    WHERE id = v_id ;

END $$

DELIMITER ;
```

- 스토어드 프로시저 실행

```
DECLARE @name VARCHAR(25) ;
```

```
DECLARE @age INT ;
```

```
CALL retrieveCustomerById ('java2', @name, @age) ;
```

```
SELECT @name, @age ;
```

● 예시

- 고객 아이디 순으로 고객 정보 목록을 조회하는 프로시저를 만들어 보자.

```
DELIMITER $$  
  
CREATE PROCEDURE retrieveCustomerList ( )  
  
BEGIN  
  
    SELECT id, name , age , grade , point  
  
    FROM customer  
  
    ORDER BY id ASC ;  
  
END $$  
  
DELIMITER ;
```

- 스토어드 프로시저 실행

```
CALL retrieveCustomerList() ;
```

● 스토어드 프로시저 변수 선언

1. 변수 선언

```
DECLARE 변수명 DATATYPE(SIZE) DEFAULT 디폴트값 ;
```

2. 변수 대입

```
SET 변수명 = 값 ;
```

ex)

```
DECLARE total INT DEFAULT 0 ;
```

```
SET total = 10;
```

※ MySQL SELECT INTO Variable syntax

```
SELECT  C1, C2, C3, ...  
INTO @V1, @V2, @V3, ...  
FROM 테이블명  
WHERE 조건 ;
```

● 스토어드 프로시저 조건문 IF ELSEIF / CASE

1. IF 문

```
IF 조건식 THEN  
    실행문;  
END IF ;
```

2) IF ELSE 문

```
IF 조건식 THEN  
    실행문 1;  
ELSE  
    실행문 2;  
END IF ;
```


3) IF ELSEIF ELSE 문

```
IF  조건식 1 THEN
    실행문 1;
ELSEIF 조건식 2 THEN
    실행문 2;
ELSE
    실행문 2;
END IF ;
```

4) CASE 문

```
CASE case_expression
    WHEN when_expression1 THEN commands ;
    WHEN when_expression1 THEN commands ;
    ...
    [ ELSE commands ; ]
END CASE;
```

● 스토어드 프로시저 반복문 - WHILE / REPEAT / LOOP

1. WHILE 문

```
WHILE 조건식 DO  
    명령문;  
END WHILE ;
```

2. REPEAT 문

- 명령문을 실행 한 후 조건을 확인한다.

```
REPEAT  
    명령문;  
    UNTIL 조건식  
END REPEAT ;
```

3. LOOP 문

- LEAVE : break 처럼 동작한다.
- ITERATE : continue 처럼 동작한다.

```
[label]: LOOP  
  
    ...  
    -- terminate the loop  
  
    IF condition THEN  
        LEAVE [label] ;  
    END IF;  
  
END LOOP;
```

● 예시

str : 2, 4, 6, 8, 10,

```
DELIMITER $$
CREATE PROCEDURE LoopDemo()
BEGIN
    DECLARE x INT;
    DECLARE str VARCHAR(255);

    SET x = 1;
    SET str = '';

    loop_label: LOOP
        IF x > 10 THEN
            LEAVE loop_label;
        END IF;

        SET x = x + 1;
        IF (x mod 2) THEN
            ITERATE loop_label;
        ELSE
            SET str = CONCAT(str, x, ',');
        END IF;
    END LOOP;

    SELECT str;

END$$
DELIMITER;

CALL LoopDemo();
```

스토어드 함수 (Stored Function)

- 스토어드 프로시저와 차이점은 리턴 값이 반드시 존재해야 한다는 것이다.
- SELECT, INSERT 등 SQL 문과 함께 사용할 수 있다.

```
DROP FUNCTION IF EXISTS function_name ;

DELIMITER $$

CREATE FUNCTION function_name ( param1, param2, ... )
    RETURN datatype
BEGIN
    SQL 명령문 ;
    RETURN 값 ;
END $$

DELIMITER ;
```

● 예시

```
DROP FUNCTION IF EXISTS customerLevel ;

DELIMITER $$

CREATE FUNCTION customerLevel (
    credit DECIMAL(10, 2)
)
RETURNS VARCHAR(30)
DETERMINISTIC
BEGIN
    DECLARE customer_level VARCHAR(30) ;

    IF credit > 50000 THEN
        SET customer_level = 'PLATINUM' ;
    ELSEIF credit >= 10000 THEN
        SET customer_level = 'GOLD' ;
    ELSE
        SET customer_level = 'SILVER' ;
    END IF;

RETURN (customer_level) ;
END $$

DELIMITER ;
```

SHOW FUNCTION STATUS WHERE db = '데이터베이스명' ;

● Calling a stored function in an SQL statement

```
SELECT customerName, customerLevel(creditLimit)
FROM customers
ORDER BY customerName;
```

트리거 (Trigger)

- 특정 테이블에 INSERT, DELETE, UPDATE 같은 DML 문이 수행될 때 데이터베이스에서 자동으로 동작한다.

● 트리거 생성

```
DELIMITER //
```

```
CREATE TRIGGER 트리거명
```

```
    {BEFORE / AFTER}    {INSERT / DELETE / UPDATE}
```

```
    ON 테이블명
```

```
    FOR EACH ROW        --- 이벤트가 행 단위로 실행된다.
```

```
BEGIN
```

```
    처리할 내용
```

```
END //
```

```
DELIMITER ;
```


● OLD / NEW 키워드

- 변경 전 또는 변경 후의 레코드(rows)는 OLD, NEW 라는 가상 줄 변수를 이용해 사용 가능하다.
- 트리거 본문 내에서 OLD.컬럼명 및 NEW.컬럼명을 사용하면 트리거의 영향을 받는 행의 열에 액세스 가능하다.

이벤트	OLD	NEW
INSERT	×	○
UPDATE	○	○
DELETEint(n)	○	×

● BEFORE / AFTER

- 트리거 실행 시점
- BEFORE : 이벤트 발생 전에 트리거 실행
- AFTER : 이벤트 발생 후에 트리거 실행

● INSERT / UPDATE / DELETE

- 트리거가 실행될 이벤트

● FOR EACH ROW - 이벤트가 발생하면 행(row) 당 한번씩 실행문을 수행한다.

● 트리거 확인

```
SHOW triggers ;
```

● 예시

- 탈퇴한 고객의 아이디, 이름, 휴대폰번호, 탈퇴일자 정보를 저장하는 트리거를 생성해 보자.

1. 삭제된 회원정보가 저장될 테이블 만들기

```
CREATE TABLE WithdrawalCustomer (  
    customer_no          VARCHAR(25)    PRIMARY KEY ,  
    customer_name        VARCHAR(25) ,  
    customer_phone       VARCHAR(25) ,  
    wthdrawal_date       DATE  
) ;
```

2. 트리거 만들기

```
DELIMITER //  
  
CREATE TRIGGER withdrawl_trigger  
    AFTER DELETE ON customer  
    FOR EACH ROW  
BEGIN  
    INSERT INTO WithdrawalCustomer (  
        customer_no, customer_name,  
        customer_phone, wthdrawal_date  
    ) VALUES (  
        OLD.customerNumber, OLD.customerName,  
        OLD.pone, CURDATE()  
    );  
END //  
  
DELIMITER ;
```

3. 삭제된 회원정보 확인하기

```
DELETE FROM customer  
  
WHERE customerName = '일길동' ;  
  
SELECT * FROM customer ;  
  
SELECT * FROM WithdrawalCustomer ;
```