파이썬 실습1

- 각 자리가 숫자**(0~9)**로만 이루어진 문자열 **S**의 가장 큰 수를 구하는 프로그램
  - 왼쪽부터 오른쪽으로 하나씩 모든 숫자를 확인
  - 숫자 사이에 '+' 또는 '*' 연산자를 넣어 결과적으로 만들어 질 수 잇는 가장 큰 수를 구함
  - 모든 연산은 왼쪽에서부터 순서대로 이루어진다는 가정
  - 입력조건
    - 문자열 s의 길이: 1 ~ 20

```
enter numeric 02984
02984의 최대 연산값은 576
```

```
enter numeric 567
567의 최대 연산값은 210
```

파이썬 실습2

- 안테나 설치 문제
  - 일직선상의 마을에 여러 채의 집이 있는 경우
  - 한 집에 한 개의 안테나를 설치하기로 결정
  - 효율성을 위해 안테나로부터 모든 집까지의 거리의 총합이 최소가 되도록 설치
  - 안테나는 집이 위치한 곳에만 설치 할 수 있으며
  - 논리적으로 동일한 위치에 어려 개의 집이 존재하는 것이 가능
  - 입력값은 집의 댓수(1 ~ 200,0000)와 각 위치값(1 ~ 100,000)
  - 출력은 안테나를 설치할 수 있는 위치값(여러 개의 값이 도출되면 가장 작은 값 출력)

```
enter num of house >> 4
enter location >> 5 1 7 9
house to install antenna 5
```

파이썬 실습3

치킨 배달 문제

- A 도시의 주민들은 치킨을 매우 좋아해서 각 집과 가장 가까운 거리에 위치한 치킨집과의 거리를 치킨거리라고 함
- 각 집들은 치킨거리를 가지고 있으며 도시의 치킨거리는 모든 집의 치킨거리를 합한 값임
- 거리는 (r1,c1)과 (r2,c2) 라면 | r1-r2| + |c1-c2|로 계산
- 도시의 크기는 N * N으로 표시되고, 각 구간은 1*1로 나누어져 있으며 집은 1, 치킨집은 2, 0은 빈 공간을 의미함(집의 개수는 2N을 넘지않고 적어도 1개 존재)
- 치킨집은 모드 한 프랜차이즈로 오랜 연구끝에 이 도시에서 가장 수입을 많이 낼 수 있는 치킨집의 개수가 M이라는 사실을 알아냄
- 입력은 도시사이즈 N(2~50), 치킨집 최대 수 M(1~13), 도시정보 이며
- 출력은 도시의 치킨 거리 최솟값

파이썬 실습3

- 치킨집에서 **M** 개를 고르는 조합을 고려
  - 최대 13개 중에서 M 선택
  - 조합(combination), $_{13}C_M$ 의 값은 100,000을 넘지 않음
  - 집의 개수 또한 최대 100개이므로 경우의 수를 모두 고려해도 되는 문제

```
enter map and chicken house >> 5 3
enter map type >> 0 0 1 0 0
enter map type >> 0 0 2 0 1
enter map type >> 0 1 2 0 0
enter map type >> 0 0 1 0 0
enter map type >> 0 0 0 0 2
치킨거리의 최솟값은 5
```

```
enter map and chicken house >> 5 2
enter map type >> 0 2 0 1 0
enter map type >> 1 0 1 0 0
enter map type >> 0 0 0 0 0
enter map type >> 2 0 0 1 1
enter map type >> 2 2 0 1 2
치킨거리의 최솟값은 10
```

```
if __name__ == "__main__":
    n,m = map(int, input("enter map and chicken house >> ").split())
    result = chicken_house(n,m)
    print(f'치킨거리의 최솟값은 {result}')
```

RDB-mysql

데이터베이스 구축: 고급SQL

2020.09. 14. 월요일

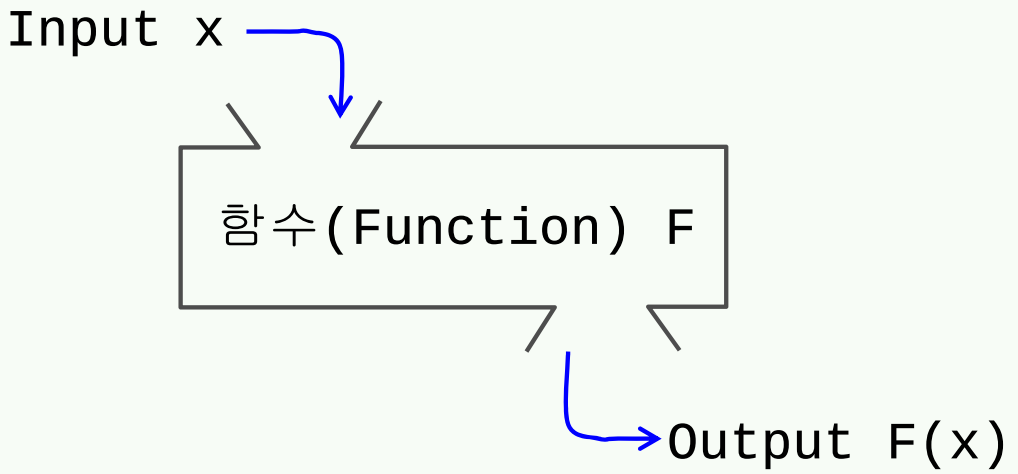
최회련

En-CORE

Data Science Edu.

SQL 내장함수

- 함수의 개념을 사용
  - 함수는 특정 값이나 열의 값을 입력 받아 그 값을 계산 후 해당 결과값을 돌려줌



  - 함수의 종류
    - DBMS가 제공하는 내장 함수(built-in function)
    - 사용자가 작성하는 사용자 정의 함수(user-defined function)

SQL - MySQL

- **MySQL**에서 사용하는 내장 함수**(built-in function)**
  - SQL 내장함수는 상수나 속성 이름을 입력 값으로 받아 단일 값을 결과로 반환
  - 모든 내장함수는 최초에 선언될 때 유효한 입력 값을 받아야 함
  - SQL 내장함수는 SELECT, WHERE, UPDATE 절 등에서 모두 사용 가능

```
SELECT  ....  함수명(인자1,인자2,...)
FROM    테이블
WHERE   ....  열이름=함수명(인자1,인자2,...);

UPDATE 테이블
SET       .... 여이름 = 함수명(인자1,인자2,...);
```

SQL - 내장함수

- **MySQL**에서 제공하는 주요 내장함수

dev.mysql.com/doc/refman/8.0/en/functions.html

구분		함수 이름
단일행 함수	숫자함수	ABS,CEIL,COS,EXP,FLOOR,LN,LOG,MOD,POWER,RAND,ROUND,SIGN,TRUNCATE
	문자함수 (문자 반환)	CONCAT,LEFT,RIGHT,LOWER,UPPER,LPAD,RPAD,LTRIM,RTRIM, REPLACE,REVERSE,RIGHT,SUBSTR,TRIM
	문자함수 (숫자 반환)	ASCII,INSTR,LENGTH
	날짜/시간함수	ADDDATE,CURRENT_DATE,DATE,DATEDIFF,DAYNAME,LAST_DAY, SYSDATE,TIME
	변환함수	CAST,CONVERT,DATE_FORMAT,STR_TO_DATE
	정보함수	DATABASE,SCHEMA,ROW_COUNR,USER,VERSION
	NULL 관련 함수	COALESCE,ISNULL,IFNULL,NULLIF
집계함수		AVG,COUNT,MAX,MIN,STD,STDDEV,SUM
윈도우함수(또는 분석함수)		CUME_DIST,DENSE_RANK,FIRST_VALUE,LAST_VALUE,LEAD,NTILE,RANK, ROW_NUMBER

SQL - 내장함수

- 숫자 함수

함수	의미 및 사용법
ABS(숫자)	숫자의 절댓값을 계산 ABS(-4.5) → 4.5
CEIL(숫자)	숫자보다 크거나 같은 최소의 정수 CEIL(4.1) → 5
FLOOR(숫자)	숫자보다 작거나 같은 최소의 정수 FLOOR(4.1) → 4
ROUND(숫자, m)	숫자의 반올림, m은 반올림 기준 자릿수 ROUND(5.36, 1) → 5.4
LOG(n, 숫자)	숫자의 자연로그 값을 반환 LOG(10) → 2.30259
POWER(숫자, m)	숫자의 m 제곱 값 계산 POWER(2,3) → 8
SQRT(숫자)	숫자의 제곱근 계산(숫자는 양수만) SQRT(9.0) → 3.0
SIGN(숫자)	숫자가 음수면 -1, 양수면 1, 0이면 0 SIGN(3.45) → 1

SQL - 내장함수

숫자 함수 예제

-78과 +78의 절댓값

```
SELECT   ABS(-78), ABS(+78);
FROM    Dual;
```

4.875를 소수 첫째 자리까지 반올림한 값

```
SELECT ROUND(4.875, 1);
FROM   Dual;
```

Dual Table:
- 오라클 자체에서 제공되는 테이블
- 간단하게 함수를 이용해서 계산 결과값을 확인할 때 사용하는 테이블
- 즉, sys사용자가 소유하는 표준 테이블
- 오직 한행, 한 컬럼을 담고 있는 dummy 테이블

고객별 평균 주문 금액을 백 원 단위로 반올림한 값

```
SELECT    custid '고객번호', ROUND(SUM(saleprice)/COUNT(*), -2) '평균금액'
FROM      Orders
GROUP BY custid;
```

	고객번호	평균금액
▶	123	23800
	124	18500
	125	21500
	126	28000

SQL - 내장함수

문자함수

반환	함수	내용
문자값 반환함수 s:문자열 c:문자 n: 정수 k: 정수	Concat(s1,s2)	두 문자열을 연결, concat('선능','출판사') →'선능출판사 '
	Lower(s)	문자열을 소문자로 변환
	Lpad(s,n,c)	문자열의 왼쪽부터 지정한 자릿수까지 지정한 문자로 채움 Lpad('korea', 10, '*') → '*****korea'
	Replace(s1,s2,s3)	문자열의 지정한 문자를 원하는 문자로 변경 Replace('길동', '길','희') →'희동'
	Rpad(s,n,c)	
	Substr(s,n,k)	문자열의 지정된 자리에서부터 지정된 길이만큼 잘라서 반환 Substr('abcdefg' 3,4) → 'cdef'
	Trim(c from s)	문자열의 양쪽에서 지정된 문자 삭제(문자열만 입력 시 공백제거) Trim('=' from '==Happy==') → 'Happy'
	Upper(s)	
숫자값 반환함수	ASCII(c)	
	Length(s)	문자열의 byte 반환, 알파벳은 1byte, 한글은 3byte
	Char_Length(s)	문자열의 문자 수 반환, char_length('한글') → 2

SQL - 내장함수

- 문자 함수 실습

도서제목 파이썬을 자바로 변경 후 도서 목록 출력

```
SELECT    bookid, REPLACE(bookname, '파이썬 ' ,  ' 자바') bookname, publisher, price
FROM      Book;
```

출판사 이름에 '출판사 ' 를 추가 하기

```
update book
set publisher = concat(publisher, '출판사')
where bookid;
```

고객 중 같은 지역에 사는 사람이 몇 명이나 되는지 확인

```
Select SUBSTR(address, 1, 2) as '지역', Count(*) '인원'
From Customer
Group by SUBSTR(address, 1, 2);
```

SQL - 내장함수

- 시간/날짜 함수
  - MySQL은 WHERE 절 없이 SELECT 만 사용 가능

SQL 문	실행 결과
SELECT now();	2020-06-17 10:16:20
SELECT current_date();	2020-06-17
SELECT now()+0;	20200617101620
SELECT current_date() + 0;	20200617
SELECT DATE_FORMAT(sysdate(),'%y%m%d:%H/%i/%s')	200617:10/16/20
SELECT DATEDIFF(date1,date2) ex)SELECT DATEDIFF('2020-06-19','2020-06-16')	2 (날짜의 차이 반환)
SELECT ADDDATE(sysdate(), INTERVAL 3 day)	2020-06-20 10:16:20
SELECT SEC_TO_TIME(3000)	00:50:00
SELECT DAYOFNAME(now())	Wednesday
SELECT DAYOFWEEK(now())	수요일:3

SQL - 내장함수

시간/날짜함수

Format의 주요 지정자

인자	설명
%w / %W	요일 순서**(0~6,Sunday=0) /** 요일**(Sunday ~ Saturday)**
%a	요일의 약자**(Sun ~ Sat)**
%d	**1**달 중 날짜**(00 ~ 31)**
%j	**1**년 중 날짜**(001 ~ 366)**
%h / %H	**12**시간**(01 ~ 12) / 24**시간**(00 ~ 23)**
%i	분**(0 ~ 59)**
%m / %M	월 순서**(01~12, January=0) /** 월 이름**(January~ December)**
%b	월 이름 약어**(Jan ~ Dec)**
%s	초**(0 ~ 59)**
%y / %Y	**4**자리 연도의 마지막 **2**자리 **/ 4**자리 연도

SQL - 내장함수

주문일로부터 3일 후 매출을 확정한다. 각 주문의 확정일자

```
SELECT    orderid '주문번호', orderdate '주문일',
          ADDDATE(orderdate, INTERVAL 3 DAY) '확정'
FROM      Orders;
```

2020년 9월 9일에 주문받은 도서의 주문번호, 주문일, 고객번호,
단, 주문일은 '%Y-%m-%d' 형태로 표시한다.

```
SELECT    orderid '주문번호', DATE_FORMAT(orderdate, '%Y-%m-%d %W') '주문일',
          custid '고객번호', bookid '도서번호'
FROM      Orders
WHERE     orderdate ='2020-09-09';
```

DBMS 서버에 설정된 현재 날짜와 시간 및 요일을 확인

```
SELECT    SYSDATE() systme1, DATE_FORMAT(SYSDATE(), '%y-%m-%d %M %W %h:%i') System2;
```

NULL 값

- **NULL 값**
  - 아직 지정되지 않은 값
  - NULL은 0 또는 공백문자 등과는 다른 특별한 값
  - NULL 값은 비교 연산자로 비교가 불가능함
  - NULL 값의 연산을 수행하면 결과 역시 NULL 값으로 반환
- **집계 함수 사용 시 주의점**
  - NULL + 숫자의 결과는 NULL
  - 집계 함수 계산 시 NULL이 포함된 행은 집계 대상에서 제외됨
  - 해당되는 행이 하나도 없는 경우, SUM, AVG 함수의 결과는 NULL이며, COUNT 함수의 결과는 0 임

NULL 값 처리

- **Profit** 테이블 작성

pro_id	sales	tax	net	sales_date
1	50000	0.07	0	2020-09-07
2	80000	0.08	0	2020-09-08
3	100000	0.1	0	2020-09-09
4	60000	0.07	0	2020-09-10
5	150000	0.11	0	2020-09-11
6	NULL	0.08	0	2020-09-12

```
SELECT  sales+100
FROM    profit
WHERE   pro_id=6;
```

```
SELECT   SUM(sales), AVG(sales), COUNT(*), COUNT(sales)
FROM    profit;
```

	SUM(sales)	AVG(sales)	COUNT(*)	COUNT(sales)
▶	440000	88000.0000	6	5

NULL 값 처리

- **NULL** 값을 확인하는 방법
  - IS NULL, IS NOT NULL
  - NULL값을 찾는 경우에는 = 가 아닌 IS NULL 사용
  - NULL이 아닌 값을 찾는 경우는 IS NOT NULL 사용
  - 예시

```
select *
from profit
where sales is null;
```

  - 다음 결과는?

```
select *
from profit
where sales = '';
```

NULL 값 처리

IFNULL

- NULL 값을 다른 값으로 대치하여 연산하거나 다른 값으로 출력
- IFNULL(속성, 값) : 속성값이 NULL 이면 다른 값으로 변경

판매가격, 세금, 판매날짜가 포함된 profit 테이블 내용 보기,
단, 판매가격이 없는 경우는 '판매가격미정 ' 으로 표시

```
select IFNULL(sales,'판매가격미정') sales, tax, sales_date
from profit
```

행번호 출력

- 내장 함수는 아니지만 자주 사용되는 문법
- **mySQL**에서 변수는 이름 앞에 @ 기호를 붙이고, 치환문에서는 **SET**과 **:=** 기호를 사용함, 전역변수
- 자료를 일부분만 확인하여 처리할 때 유용

Customer에서 고객번호, 이름, 전화번호를 앞의 두 명만 보이시오.

```
set @seq := 0;

select (@seq := @seq + 1) '순번', custid, name, phone
from Customer
where @seq < 3;
```

순번	custid	name	phone
1	123	길동	555-5555
2	124	영희	555-7777
3	125	철수	333-3333

데이터 형식과 형 변환

CAST / CONVERT

- 사용 형식의 차이를 제외하고 거의 비슷한 기능을 가짐

```
cast (expression   AS 데이터형식 [(길이)])

convert (expression,  데이터형식 [(길이)])
```

- 사용 예

```
Use encored;
Select avg(price) as '평균 도서 가격' from book;
```

  - 가격임으로 정수형으로 표현하고 싶은 경우

```
Select cast(avg(price) as INT) as 'average price' from book;
또는

Select convert(avg(price), INT) as 'average price' from book;
```

데이터 형식과 형 변환

- 암시적인 형 변환
  - CAST 또는 CONVERT 함수를 사용하지 않는 형 변환
  - 사용 예

```
select '100' + '200' as sum; -- 숫자로 변환되어 계산, 300의 결과
select concat('100', '200'); -- 문자열 연결 함수 concat 사용
select concat(100,'200'); -- 숫자+문자는 숫자가 문자로 변환되어 연산
select 1 + '2mega' ; -- 정수 2로 변환되어 연산
select 0 = 'mega'; -- 문자는 0으로 변환
```

  - 다음의 결과?

```
select 1 + '3.5encore';
select 3 + 'encore4.2';
select 0.0  >  '4mega';
select 0.0 > '-3mega';
```

제어 흐름 함수

- 프로그램의 흐름을 제어하는 함수
  - IF, IFNULL, NULLIF, CASE ~ WHEN ~ ELSE ~ END

- **IF**
  - 수식이 참 또는 거짓인지 결과에 따라서 실행문이 선택되어 실행

```
select if(100>200, '참','거짓') as '100>200';
```

Safe Mode Error 발생 시 safe mode 끄기로 오류발생을 처리할 수 있음
(1) set sql_safe_updates=0;
(2) Workbench에서 [Edit] → [Preferences]에서 'SQL Editor' 항목 중
Safe Updates (reject Updates and Deletes ~) 속성 체크를 삭제

Error Code: 1175. You are using safe update mode ~~~~

제어 흐름 함수

IFNULL / NULLIF

- ifnull(수식1, 수식2)
  - 수식1이 NULL이 아니면 수식1이 반환, 수식1이 NULL이면 수식2가 반환
- nullif(수식1,수식2)
  - 수식1과 수식2가 같으면 NULL 반환하고, 다르면 수식1을 반환

```
select ifnull(10*20, NULL);
```

```
select nullif(10*20,null);
select nullif(10*20,10*20);
```

제어 흐름 함수

- **CASE ~ WHEN ~ ELSE ~ END**
  - case 는 연산자로 분류
  - 다중 분기에 사용될 수 있음
  - case 연산자 옆의 값에 따라 When 옆의 값으로 분기하여 Then 뒤의 식을 실행
  - When의 값이 해당사항이 없는 경우는 else 가 실행 됨

```
SELECT CASE 10
                    WHEN 1 THEN ‘one’
                    WHEN 5 THEN ‘five’
                    WHEN 10 THEN ‘ten’
                    ELSE ‘nothing’
          END AS ‘CASE EXAM’;
```

피벗(pivot) 구현

- 한 열에 포함된 여러 값을 출력하고 이를 여러 열로 변환
- 테이블 반환 식을 회전하고 필요하면 집계까지 수행하는 것
- **sum, if, group by** 를 활용함

	uName	봄	여름	가을	겨울	합계
▶	길동	40	14	25	32	111
	희동	0	65	0	20	85

	uName	season	amount
▶	길동	겨울	10
	희동	여름	15
	길동	가을	25
	길동	봄	3
	희동	겨울	20
	길동	봄	37
	길동	여름	14
	길동	겨울	22
	희동	여름	50

```
select uName,
sum(if(season = '봄',amount,0)) as '봄',
sum(if(season = '여름',amount,0)) as '여름',
sum(if(season = '가을',amount,0)) as '가을',
sum(if(season = '겨울',amount,0)) as '겨울',
sum(amount) as '합계'
from pivottest
group by uName;
```

피벗(pivot) 구현

- **Table** 생성

```
CREATE TABLE pivottest (
uName  CHAR(10) NOT NULL,
season  CHAR(10) NULL,
amount  INT NULL)
DEFAULT CHARACTER SET = utf8;
```

- 데이터 입력

```
INSERT INTO pivottest (uName,season,amount)
VALUES('길동','겨울',10), ('희동','여름',15),('길동','가을',25),('길
동','봄',3),('희동','겨울',20), ('길동','봄', 37), ('길동','여름',14),
('길동','겨울',22), ('희동','여름',50);
```

뷰(View)

- 하나 이상의 테이블을 합하여 만든 가상의 테이블
- 합한다는 의미는 **SELECT** 문을 통한 최종 결과를 의미
- 물리적 존재인 기본 테이블과는 달리 논리적으로만 존재함
- 질의문을 통한 결과를 하나의 가상 테이블로 정의하여 실제 테이블처럼 사용할 수 있도록 만든 데이터베이스 개체
  - 뷰 생성 시 기반이 되는 물리적인 테이블을 기본 테이블이라 함
  - 일반적으로 기본 테이블을 기반으로 만들어지지만, 다른 뷰를 기반으로도 생성 가능 함
  - 뷰를 통해 기본 테이블의 내용을 쉽게 검색할 수 있지만, 기본 테이블의 내용 변경 작업은 제한적으로 이루어짐

뷰(View)

- 뷰 생성 - **ORDER BY** 를 사용할 수 없음

```
CREATE VIEW 테이블이름[(속성_리스트)]
AS SELECT 문
[WITH CHECK OPTION];
```

  - 등급이 vip인 우수고객의 정보를 출력하는 VIEW 생성

```
CREATE VIEW VIP_Customer(custid,name,age,address)
AS SELECT custid, name, age, address
    FROM   Customer
    Where  grade = 'vip'

WITH CHECK OPTION;
```

- 뷰 삭제
  - DROP VIEW 뷰_이름

뷰(View)

뷰 활용의 장점

- 질의문을 좀 더 쉽게 작성할 수 있음
  - 특정 조건을 만족하는 투플들로 뷰를 미리 만들어 놓으면 사용자가 WHERE 절 없이 뷰를 검색해도 특정 조건을 만족하는 데이터 검색이 가능
  - GROUP BY, 집계함수, 조인 등을 이용해 뷰를 생성하면 SELECT와 FROM 절만으로 원하는 결과를 쉽게 얻을 수 있음
- 데이터의 보안 유지에 도움
  - 여러 사용자의 요구에 맞는 다양한 뷰를 미리 정의해두고 사용자가 자신에게 제공된 뷰를 통해서만 데이터에 접근하도록 권한 설정을 하면 보안 유지에 도움이 됨
- 데이터를 좀 더 편리하게 관리할 수 있음

뷰(view)

- 실습

<기본테이블 book>

bookid	bookname	publisher	price
12	파이썬	태릉	15000
13	딥러닝	선릉	20000
14	하둡	태릉	17000
15	엘리스	디자인	22000
16	R	선릉	23000
17	알고리즘	태릉	19000
18	자바	한미	26000
19	파이토치활용	태릉	27000
20	데이터베이스	한미	25000

- Book 테이블에서 View 테이블 생성
  - 책 금액이 전체 평균 이상의 책을 보유한
    출판사 및 책이름, 책아이디, 가격 출력

```
create view mainBook
as select bookid, bookname, publisher,price
from book
where price > (select avg(price) from book)
order by publisher;

select * from mainBook;
```

- 생성된 publisher에 (’21’, ‘케라스‘,’한미‘,’28000’)의 새로운 레코드 추가

```
insert into mainBook values (21,'케라스','한미출판사',28000);
```

→ 입력 오류 발생

뷰(view)

- 뷰의 수정 시 기본 테이블도 함께 수정됨
- 뷰의 수정 오류인 경우
  - 기본 테이블의 기본키를 구성하는 속성이 없는 경우
  - 기본 테이블에 없는 속성이 있는 경우
    - 집계함수 사용 등
  - Distinct, Group by 를 포함하여 뷰생성 시
  - 여러 개의 테이블을 조인하여 뷰생성 시
    - Join 뿐 아니라, 부속질의어도 포함

뷰(view)

- **Shoppingmall DB**를 이용하여 **View table** 생성
  - Member, product, order_prod 테이블 구성

	Field	Schema	Table	Type	Character Set	Display Size
1	memberid	shoppingmall	member	CHAR	utf8mb4	8
2	membername	shoppingmall	member	CHAR	utf8mb4	15
3	membertel	shoppingmall	member	CHAR	utf8mb4	15

	Field	Schema	Table	Type	Character Set	Display Size
1	productid	shoppingmall	product	INT	binary	11
2	productname	shoppingmall	product	VARCHAR	utf8mb4	20
3	productprice	shoppingmall	product	INT	binary	11
4	company	shoppingmall	product	VARCHAR	utf8mb4	10
5	p_member	shoppingmall	product	CHAR	utf8mb4	8

	Field	Schema	Table	Type	Character Set	Display Size
1	orderid	shoppingmall	orders_p	INT	binary	11
2	order_amount	shoppingmall	orders_p	INT	binary	11
3	order_cust	shoppingmall	orders_p	CHAR	utf8mb4	8
4	order_prod	shoppingmall	orders_p	INT	binary	11
5	order_date	shoppingmall	orders_p	DATE	binary	10

  - View table, order_view 생성

	Field	Schema	Table	Type	Character Set	Display Size
1	고객아이디	shoppingmall	order_view	CHAR	utf8mb4	8
2	고객이름	shoppingmall	order_view	CHAR	utf8mb4	15
3	주문제품	shoppingmall	order_view	INT	binary	11
4	주문수량	shoppingmall	order_view	INT	binary	11

뷰(view)

- **View Table 생성시**
  - Create View 일 경우는 기존의 동일한 뷰가 있으면 오류발생
  - Create or replace view 는 기존 뷰에 덮어쓰는 효과로 오류가 발생하지 않음
    - Drop view 와 create view를 연속 쓴 효과

```
create or replace view order_view
as
select C.memberid as '고객아이디', C.membername as '고객이름',
O.order_prod as '주문제품',O.order_amount as '주문수량‘
from member C
inner join orders_p O
on C.memberid = O.order_cust;


Select * from order_view;
Describe order_view; -- order_view의 정보 확인
Show create view order_view; -- view의 source code 확인
```

뷰(view)

- 주문 날짜가 **2020-09-12**일 이후의 주문정보에 대한 **view** 생성

```
create or replace view order_date
as
select * from orders_p
where order_date > '2020-09-12‘
with check option;
```

  - 2020-09-10일의 주문 정보 추가
    - Insert into order_date values(6, 7 ,’bb’, 125, ‘2020-09-10’); -- 오류 발생

```
insert into order_date values(6, 7, 'bb', 125, '2020-09-14');
select * from order_date;
Select * from orders_p;
```

RDB-mysql

데이터베이스 구축: 프로그래밍

2020.09. 14. 월요일

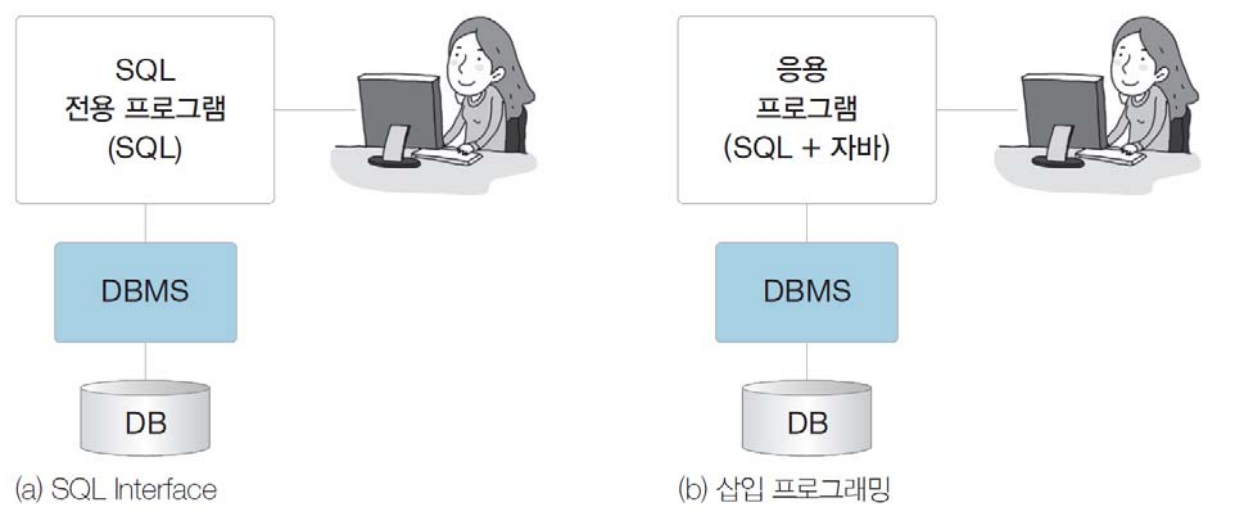
최회련

En-CORE

Data Science Edu.

데이터베이스 프로그래밍 개념

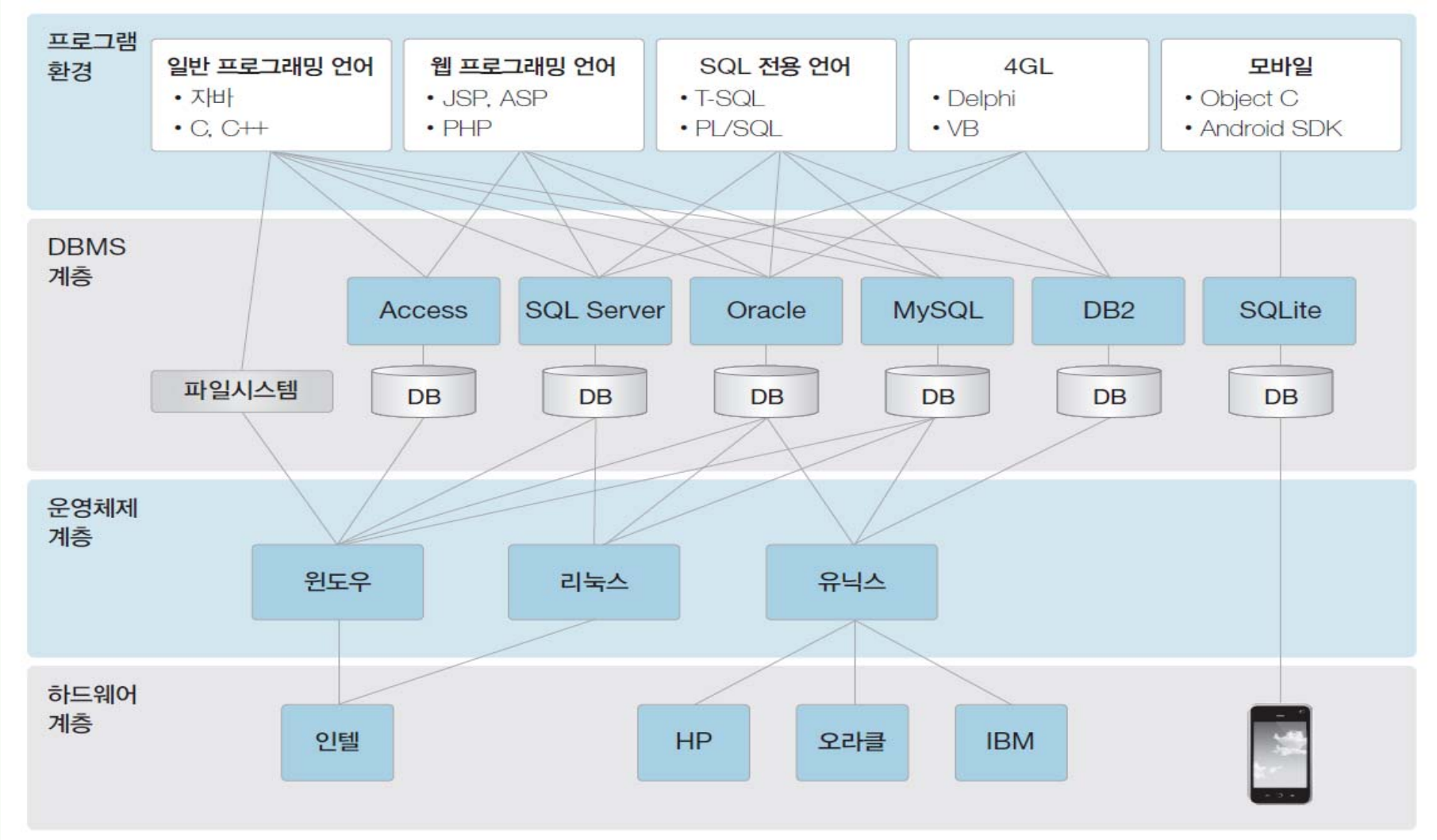
- 프로그래밍
  - 프로그램을 설계하고 소스코드를 작성하여 디버깅하는 과정
- 데이터베이스 프로그래밍
  - DBMS에 데이터를 정의하고 저장된 데이터를 읽어와 데이터를 변경하는 프로그램을 작성하는 과정
  - 데이터베이스 언어인 SQL을 포함



그림참조: MySQL로 배우는 데이터베이스 개론과 실습 , 한빛아카데미

데이터베이스 프로그래밍의 개념

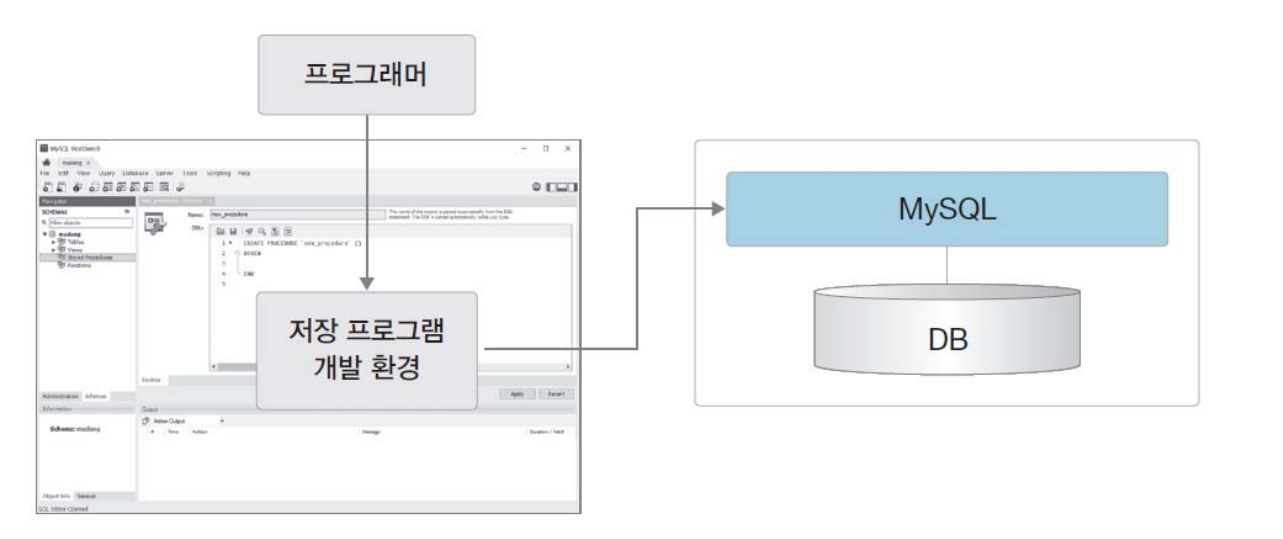
- **DBMS** 플랫폼과 데이터베이스 프로그래밍의 유형



그림참조: MySQL로 배우는 데이터베이스 개론과 실습 , 한빛아카데미

저장 프로그램(Stored Program)

- **MySQL**에서 제공되는 프로그래밍 기능
- 쿼리문의 집합으로 어떠한 동작을 일괄 처리하기 위한 용도로 사용
- 자주 사용되는 일반적인 커리를 모듈화 시켜 필요할 때마다 호출하여 사용
- 스토어드 프로시저도 데이터베이스의 개체 중 한 가지
  - 즉 테이블처럼 각 데이터베이스 내부에 저장이 됨



그림참조: MySQL로 배우는 데이터베이스 개론과 실습 , 한빛아카데미

저장 프로그램(Stored Program)

- 스토어드 프로그램은
  - 스토어드 프로시저
  - 스토어드 함수
  - 트리거
  - 커서 등이 있음
- 스토어드 프로시저(stored procedure)
  - 형식

```
DELIMITER $$
CREATE PROCEDURE 프로시저이름 (IN 또는 OUT 파라미터)
BEGIN
    SQL 프로그램 코딩...

END $$
DELIMITER ;

CALL 프로시저이름();
```

저장 프로그램(Stored Program)

- 프로시저 정의는 **CREATE PROCEDURE** 문을 사용
- 정의 방법
  - 선언부와 실행부(BEGIN – END)로 구성
  - 선언부는
    - 변수, 매개변수 선언
    - 매개변수(parameter variable)는 저장 프로시저가 호출 될 때 그 프로시저에 전달되는 값
    - 변수는 저장 프로시저나 트리거 내에서 사용되는 값
  - 실행부는 프로그램 로직 구현
  - 주석문은
    - 한 줄 주석 -- 사용
    - 여러 줄 주석 /* */ 사용

SQL 프로그래밍

- 삽입 작업 프로시저
  - 프로시저를 이용하여 데이터 삽입 작업의 효율성을 높임

```
drop procedure if exists insertBook;
delimiter //
create procedure insertBook(
in mybookid INT,              -- procedure 의 입력 매개변수 선언
in mybookname varchar(30),    -- OUT 는 출력 매개변수 선언, select ~ into 구문과 사용
in mypublisher varchar(45),
in myprice  int,in myamount INT)
begin
        insert into book(bookid, bookname,publisher,price,amount)
        values(mybookid,mybookname,mypublisher,myprice,myamount);
end //
delimiter ;
```

SQL 프로그래밍

- 제어문을 사용하는 프로시저
  - 어떤 조건에서 어떤 코드가 실행되어야 하는지를 제어하기 위한 문법
  - 절차적 언어의 구성요소를 포함함

구문	의미	문법
DELIMITER	구문 종료 기호 설정	DELIMITER {기호}
BEGIN – END	프로그램 문의 블록화 중첩 가능	BEGIN {SQL 문} END
IF – ELSE	조건의 검사 결과에 따라 문장 선택 수행 시행	IF <조건> THEN {SQL문} [ELSE {SQL 문}] END IF;
LOOP	LEAVE 문을 만나기 전까지 LOOP 문 수행	[label:] LOOP {SQL 문 | LEAVE [label]} END LOOP

SQL 프로그래밍

구문	의미	문법

WHILE	조건이 참일 경우 WHILE 문의 블록 실행	WHILE <조건> DO
		{SQL 문 | BREAK |
		CONTINUE}
		END WHILE

REPEAT	조건이 참일 경우 REPEAT 문의 블록 실행	[label:] REPEAT {SQL 문 | BREAK| CONTINUE]} UNTILE <조건> END REPEAT [label:]
RETURN	프로시저 종료 상태값을 반환	RETURN [<식>]

SQL 프로그래밍

- **Shoppingmall DB**에서 고객 **id** 로 주문 상황을 확인하는 **procedure** 작성

```
drop procedure if exists orderProc;
delimiter //
create procedure orderProc(
IN mem_id Varchar(8))
begin
        select M.membername, O.order_prod, O.order_amount,   O.order_date
        from member M inner join orders_p O on M.memberid = O.order_cust
        where M.memberid = mem_id;
end //
delimiter ;

call orderProc('aa');
```

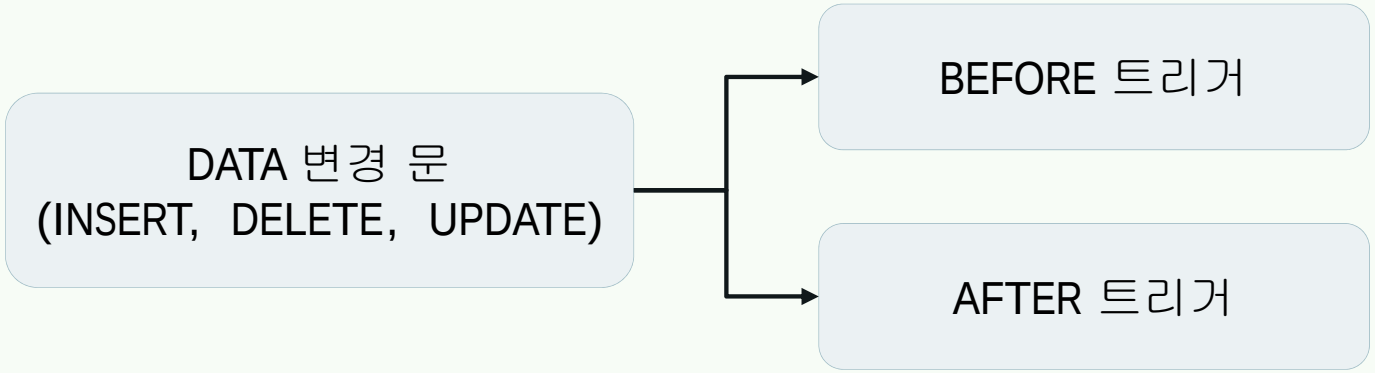
SQL 프로그래밍

- 커서를 사용하는 프로시저
  - 커서(cursor)는 실행 결과 테이블을 한 번에 한 행씩 처리하기 위하여 테이블의 행을 순서대로 가리키는데 사용함
  - 커서관련 키워드

키워드	의미
Cursor <cursor 이름> IS <커서 정의>	커서 생성
OPEN <cursor 이름>	커서의 사용을 시작
FETCH <cursor 이름> INTO <변수>	행 데이터를 가져옴
CLOSE <cursor 이름>	커서의 사용을 종료

트리거(Trigger)

- 데이터의 변경(insert, delete, update) 문이 실행될 때 자동으로 따라서 실행되는 프로시저



SQL 프로그래밍

- 사용자 정의 함수(function)
  - 수학의 함수와 마찬가지로 입력된 값을 가공하여 결과 값을 반환
  - 도서 판매 시 발생되는 이익을 계산하는 사용자 정의 함수
  - 사용 형식

```
DELIMITER <기호>
CREATE FUNCTION 함수이름 (IN 또는 OUT 파라미터)
RETRUNS <도메인 자료형>
BEGIN
    SQL 프로그램 코딩...

RETRUN <반환될 파라미터>
END <기호>
DELIMITER ;
```

  - 함수 생성 시 ERROR 1418 (HY000): This function has none of ~ 오류 발생하면
    - mysql> SET global log_bin_trust_function_creators = 1; 설정함

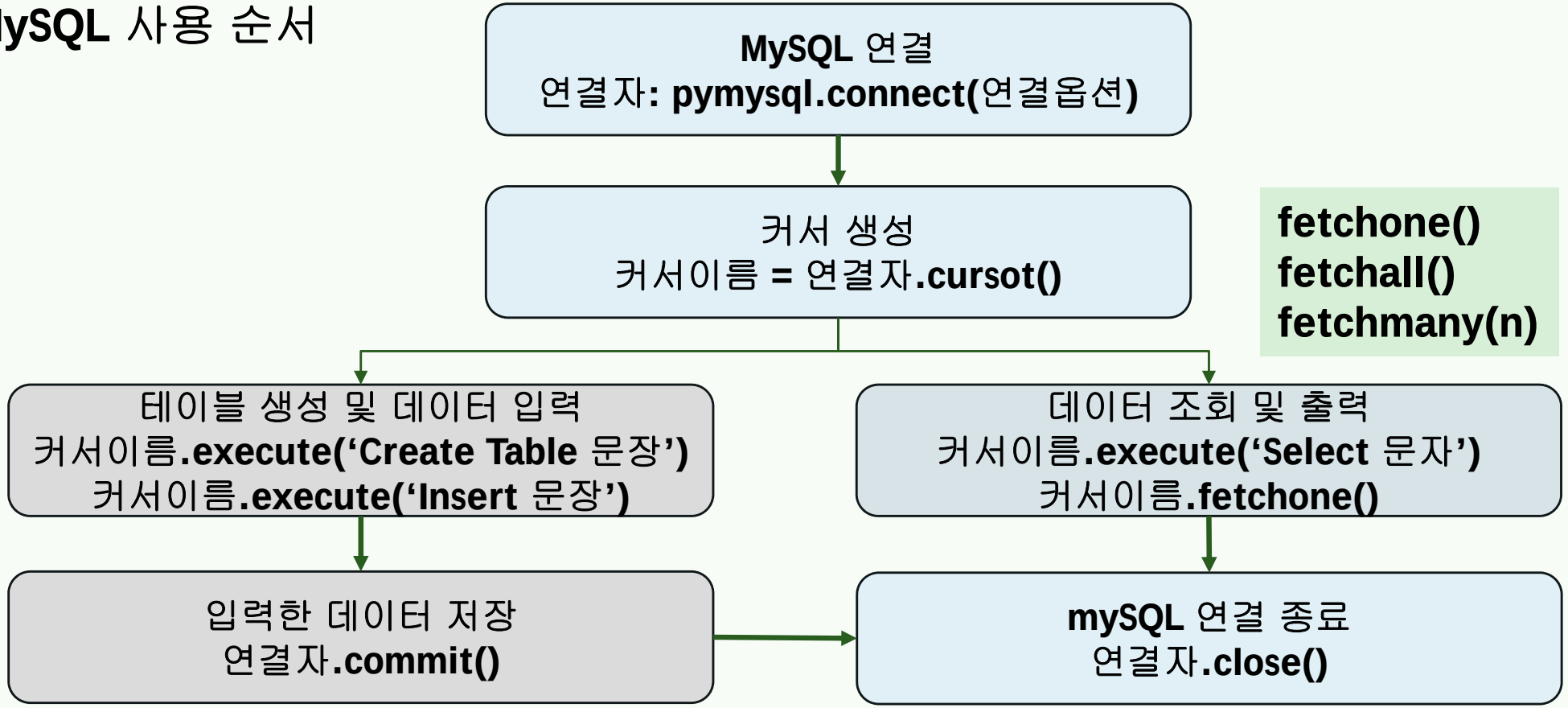
SQL 프로그래밍

- 프로시저, 트리거, 사용자 정의함수의 특징

구분	프로시저	트리거	사용자정의함수
공통점	**Stored procedure (저장 프로시저)**		
정의 방법	**Create PROCEDURE**	**Create TRIGGER**	**Create FUNCTION**
호출 방법	**CALL** 문으로 직접 호출	**INSERT,DELETE, UPDATE** 문이 실행될 때 자동으로 실행	**SELECT** 문에 포함
차이점	**SQL** 문으로 할 수 없는 복잡한 로직 수행	기본값 제공, 데이터 제약 준수, **SQL** 뷰 수정, 참조무결성 작업 등을 실행	속성 값을 가공하여 반환, **SQL** 문에서 직접 사용

Python 과 연동

- 외부 라이브러리인 **pymysql** 패키지 설치
  - Pip install pymysql
  - import pymysql 문을 통해 MySQL 관련 함수를 사용
- **MySQL** 사용 순서



Python 과 연동

- **mySQL**에 구축되 데이터베이스와 연동 후, 데이터 수집 및 출력
  - 수집된 데이터의 자료형은 기본이 tuple

```
import pymysql
#mySQL connection 연결
conn = pymysql.connect(host='localhost',user='root', password='root',
                       db='shoppingmall', charset='utf8')
curs = conn.cursor()# connection으로부터 cursor 생성

sql = 'select * from book'
curs.execute(sql)
rows = curs.fetchall()
print(rows)
#print(rows[0],rows[1])



conn.close()
```

Python 과 연동

- 테이블 생성 및 데이터 입력

```
curs.execute('drop table if exists customer')
sql = ' create table customer( \
        custid INT Primary key, \
        custname Varchar(20),  \
        address varchar(30)) \
        default character set utf8 collate utf8_general_ci \
        '
curs.execute(sql)
curs.execute('show tables')
for x in curs:
    print(x)

while True:
    data1 = input('고객 ID >> ')
    if data1 == '':
        break

    data2 = input('고객 이름 >> ')
    data3 = input('고객 주소 >> ')

    sql = "insert into customer values(" + data1 +", '" + data2 +"', \
            '"+ data3 +"')"
    curs.execute(sql)
conn.commit()
conn.close()
```

Python 과 연동

Parameter placeholder

- SQL 문에서 실제 값의 입력 위치를 동적으로 처리, 즉 동적 SQL 구문 구성
- Parameter placeholder 사용
  - Placeholder는 %s 사용
  - 파이썬의 각 자료형에 대한 변환서식과는 다름
  - 문자열 또는 숫자에 상관없이 모두 %s 를 사용하며,
  - 문자열이라도 %s 를 이용부호로 감싸지 않음
  - Placeholder 는 컬럼값을 대치할 때만 사용
    - 테이블이나 기타 문장에는 placeholder를 사용할 수 없음

Python 과 연동

- **Placeholder** 를 사용한 조회

```
import pymysql

#mySQL connection 연결
conn = pymysql.connect(host='localhost',user='root', password='root',
                       db='shoppingmall', charset='utf8')
curs = conn.cursor()# connection으로부터 cursor 생성

sql = 'select * from customer where custid=%s'
curs.execute(sql, 1)

rows = curs.fetchone()
print(rows)

conn.close()
```

Python 과 연동

Dictionary cursor 사용

- Cursor 의 기본은 Array based cursor 로 레코드(row)를 튜플 형식으로 반환
- Cursor(pymysql.cursors.DictCursor) 로 사용 시
  - 레코드(row) 를 Dictionary 형식으로 반환
  - Row 데이터가 딕셔너리 형태이므로 row 사용 시
  - 컬럼의 인덱스 대신 [컬럼이름] 으로 사용할 수 있음
  - 컬럼수가 많은 경우에 유용하게 사용될 수 있음

```
curs = conn.cursor(pymysql.cursors.DictCursor)

sql = 'select * from customer where custid=%s'
curs.execute(sql, 1)

rows = curs.fetchone()
print(rows)
print(rows['custname'])
conn.close()
```

Python 과 연동

- 데이터 삽입

```
while True:
    data1 = input('고객 ID >> ')
    if data1 == '':
        break

    data2 = input('고객 이름 >> ')
    data3 = input('고객 주소 >> ')

    sql = "insert into customer values(%s,%s,%s)"
    curs.execute(sql,(data1,data2,data3) )

conn.commit()
conn.close()
```

```
val = [ (5, '수현', '서울시 은평구'),
(6, '연아', '서울시 종로구'), ]

curs.execute(sql, val)
```

Python 과 연동

- **Pandas** 모듈 적용

```
data = curs.fetchone()

pdata = pd.DataFrame(data,index=[0])
display(pdata)
```

```
import pymysql
import pandas as pd
from IPython.display import display

#mySQL connection 연결
conn = pymysql.connect(host='localhost',user='root', password='root',
                       db='shoppingmall', charset='utf8')
curs = conn.cursor(pymysql.cursors.DictCursor)# connection으로부터 cursor 생성

sql = 'select * from customer'
curs.execute(sql)

data = curs.fetchall()

pdata = pd.DataFrame(data)
display(pdata)

curs.close()
```