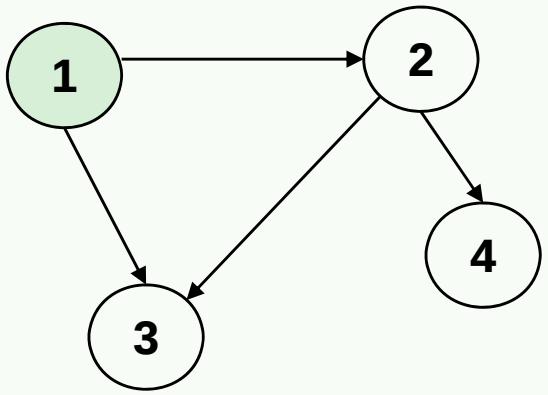
파이썬 실습1

- 특정 거리의 도시 찾기
  - 1 ~ N 번 까지의 도시와 M개의 단방향 도로로 구성
  - 모든 도로의 길이는 1로 일정, A도시에서 A도시의 길이는 0
  - 특정 도시 X로 부터 출발하여 도달할 수 있는 모든 도시 중에서 최단 거리가 정확히 K인 모든 도시의 번호를 출력하는 프로그램 작성
  - 예
    - N = 4, M = 4, K = 2, X = 1인 경우
    - K가 2인 도시는 4, 따라서 결과는 4
    - 입력 예시
      - 4 4 2 1
      - 1 2
      - 1 3
      - 2 3
      - 2 4




```
n, m, k, x = map(int, input('enter input values: ').split())
graph = [[] for _ in range(n+1)]
shortest_city(graph,n,m,k,x)
```

파이썬 실습2

- 카드 정렬하는 문제
  - N개의 숫자 카드 묶음의 각각의 크기가 주어질 때, 최소한 몇 번의 비교가 필요한지를 구하는 프로그램
  - 즉, 정렬된 두 묶음의 숫자카드가 있을 때 각 묶음의 카드 수A,B라 하면 보통 두 묶음을 합쳐서 하나로 만들 때, A + B번의 비교가 필요
  - 예를 들어 20장 카드과 30장 카드가 있다면 20 + 30 = 50 번의 비교가 필요
  - 또 다른 예로 10장, 20장 40장의 묶음이 있는 경우
    - 10 + 20 → 30 한 후에 이 결과인 30 묶음과 40을 합치면 (10 + 20) + (30 + 40) = 100번의 비교 필요
    - 만약 10 + 40을 먼저 한 후 20을 합치면, (10 + 40) + (50 + 20) = 120이 되므로 위의 해결보다 덜 효율적임
    - 숫자 카드 묶음을 입력으로 비교 횟수를 출력으로 함

파이썬 실습3

- 문자열 재정렬 문제
  - 알파벳과 숫자(0~9)로만 구성된 문자열이 입력으로 주어질 때,
  - 알파벳은 오름정렬로 출력하고 출력된 알파벳 바로 뒤에 모든 숫자를 더한 값을 이어서 출력함
  - 예
    - 입력 : K1KA5CB7　　출력 : ABCKK13

enter data (mix of alpha and number): cA8bkG9yzK
ABCGKKYZ17

RDB-mysql

데이터베이스 구축: SQL

2020.09. 11. 금요일

최회련

En-CORE

Data Science Edu.

두 개 이상의 테이블 활용한 SQL 질의 구문

- 조인(JOIN) 문법 정리

종류	형식	설명
일반	select <속성(컬럼)들> from 테이블1, 테이블2 where <조인조건> AND <검색조건>	SQL 문에서는 일반적으로 **동등조인**을 사용 - 두 가지 형식 중 하나를 적용하면 됨
	select <속성(컬럼)들> from 테이블1 **INNER JOIN** 테이블2 **ON** <조인조건> where <검색조건>	
외부	select <속성(컬럼)들> from 테이블1 {LEFT|RIGHT|FULL [OUTER]} JOIN 테이블2 **ON** <조인조건> where <검색조건>	외부조인은 From 절에 조인 종류를 작성하고 ON 키워드를 이용하여 조인조건을 명시

두 개 이상의 테이블 활용한 SQL 질의 구문

부속질의 정리

- SQL 문 내에 또 다른 SQL 문을 포함(main-query, sub-query)

가장 비싼 도서의 이름

```
SELECT      bookname
FROM        Book
WHERE       price = ( SELECT MAX(price)
                      FROM Book);
```

bookname
파이토치활용

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

두 개 이상의 테이블 활용한 SQL 질의 구문

도서를 구매한 적이 있는 고객의 이름

```
SELECT      name
FROM        Customer
WHERE       custid IN (SELECT    custid
                       FROM      Orders)
LIMIT 3; -- 출력 개수 제한, 3개까지
```

name
길동
영희
철수
희동

태릉에서 출판한 도서를 구매한 고객의 이름

```
SELECT      name
FROM        Customer
WHERE       custid IN (SELECT    custid
                       FROM      Orders
                       WHERE     bookid IN (SELECT    bookid
                                            FROM      Book
                                            WHERE     publisher= ' 태릉'));
```

name
영희
철수
길동

custid	name	address	phone
123	길동	서울	555-5555
124	영희	서울	555-7777
125	철수	부산	333-3333
126	희동	인천	777-5555
127	둘리	부산	333-2222

orderid	custid	bookid	saleprice	orderdate
1	123	13	23000	2020-09-07
2	124	17	20000	2020-09-07
3	125	14	20000	2020-09-08
4	126	18	28000	2020-09-08
5	123	16	25000	2020-09-07
6	125	13	23000	2020-09-09
7	124	12	17000	2020-09-09
8	123	17	20000	2020-09-09
9	123	20	27000	2020-09-10

두 개 이상의 테이블 활용한 SQL 질의 구문

orderid	custid	bookid	saleprice	orderdate
1	123	13	23000	2020-09-07
2	124	17	20000	2020-09-07
3	125	14	20000	2020-09-08
4	126	18	28000	2020-09-08
5	123	16	25000	2020-09-07
6	125	13	23000	2020-09-09
7	**124**	**12**	**17000**	**2020-09-09**
8	123	17	20000	2020-09-09
9	123	20	27000	2020-09-10

custid	name	address	phone
123	길동	서울	555-5555
124	**영희**	**서울**	**555-7777**
125	철수	부산	333-3333
126	희동	인천	777-5555
127	둘리	부산	333-2222

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

두 개 이상의 테이블 활용한 SQL 질의 구문

- 상관 부속질의(correlated subquery)
  - 상위 부속질의의 투플을 이용하여 하위 부속질의를 계산함
  - 상위 부속질의와 하위 부속질의가 독립적이지 않고 서로 관련을 맺고 있음

출판사별로 출판사의 평균 도서 가격보다 비싼 도서?

```
SELECT    b1.bookname
FROM      Book b1
WHERE     b1.price > (SELECT    avg(b2.price)
                      FROM      Book b2
                      WHERE  b2.publisher=b1.publisher);
```

bookname
R
자바
파이토치활용

Avg(b2.price)→(15000+17000+19000+27000)/4 : 19500원

b1

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

b2

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

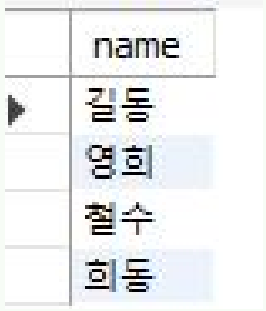
두 개 이상의 테이블 활용한 SQL 질의 구문

집합 연산

- 합집합 UNION, 차집합 MINUS, 교집합 INTERSECT

서울에서 거주하는 고객의 이름과 도서를 주문한 고객의 이름

```
SELECT    name    -- 이름 출력
FROM      Customer
WHERE     address LIKE '대한민국%'
UNION
SELECT    name
FROM      Customer
WHERE     custid IN (SELECT custid FROM Orders);
```



	name
▶	길동
	영희
	철수
	희동

{고객 이름} = {대한민국에 거주하는 고객 이름} ∪ {도서를 주문한 고객 이름}

두 개 이상의 테이블 활용한 SQL 질의 구문

- MINUS, INTERSECT 연산자가 MySQL는 없음
  - IN 또는 NOT IN 연산자 활용

[MINUS, 차집합] 부산에서 거주하는 고객의 이름에서 도서를 주문한 고객의 이름 뺀 결과
NOT IN 연산자를 사용

```
SELECT      name
FROM        Customer
WHERE address LIKE '부산%' AND
            name NOT IN (SELECT name
                         FROM       Customer
                         WHERE      custid IN (SELECT custid FROM Orders));
```

[INTERSECTION,교집합] 서울에서 거주하는 고객 중 도서를 주문한 고객의 이름
IN 연산자를 사용

```
SELECT      name
FROM        Customer
WHERE address LIKE '서울%' AND
            name IN (SELECT      name
                         FROM       Customer
                         WHERE      custid IN (SELECT custid FROM Orders));
```

두 개 이상의 테이블 활용한 SQL 질의 구문

EXITS

- 원래 단어에서 의미하는 것과 같이 조건에 맞는 투플이 존재하면 결과에 포함시킴
  - 부족질의문의 어떤 행이 조건에 만족하면 참임
- NOT EXITS는 부속질의문의 모든 행이 조건에 만족하지 않을 때만 참임

주문이 있는 고객의 이름과 주소

```
SELECT    name, address
FROM      Customer cs
WHERE     EXISTS (SELECT *
                  FROM  Orders od
                  WHERE cs.custid =od.custid);
```

	name	address
▶	길동	서울
	영희	서울
	철수	부산
	희동	인천

- 다음의 결과는?

```
select cs.name, cs.address
from customer cs, orders od
where cs.custid = od.custid;
```

```
select  cs.name, cs.addressfrom
customer cs INNER JOIN orders
odwhere cs.custid = od.custid;
```

두 개 이상의 테이블 활용한 **SQL** 질의 구문

- **EXISTS** 동작

custid	name	address	phone
123	길동	서울	555-5555
124	영희	서울	555-7777
125	철수	부산	333-3333
126	희동	인천	777-5555
127	둘리	부산	333-2222

orderid	custid	bookid	saleprice	orderdate	
1	123	13	23000	2020-09-07	**true**
2	124	17	20000	2020-09-07	**true**
3	125	14	20000	2020-09-08	**true**
4	126	18	28000	2020-09-08	**true**
5	123	16	25000	2020-09-07	
6	125	13	23000	2020-09-09	
7	124	12	17000	2020-09-09	
8	123	17	20000	2020-09-09	
9	123	20	27000	2020-09-10	

테이블 복사

- **Create Table ~~ Select 사용**
  - Create table 새로운 테이블 (select 복사할 열 from 기존 테이블)
  - 예:
    - Customer 테이블을 newCustomer 테이블에 복사
    - 전체 복사: create table newCustomer (select * from Customer);
    - 부분 열 복사: create table newCustomer (select custid, name from Customer);
  - 복사 시 Primary key 나 Foreign Key의 속성은 복사되지 않음
    - Alter Table 의 명령으로 속성 부여

```
ALTER TABLE newCustomer
ADD PRIMARY KEY (`custid`);
```

RDB-mysql

데이터베이스 구축: 고급SQL

2020.09. 11. 금요일

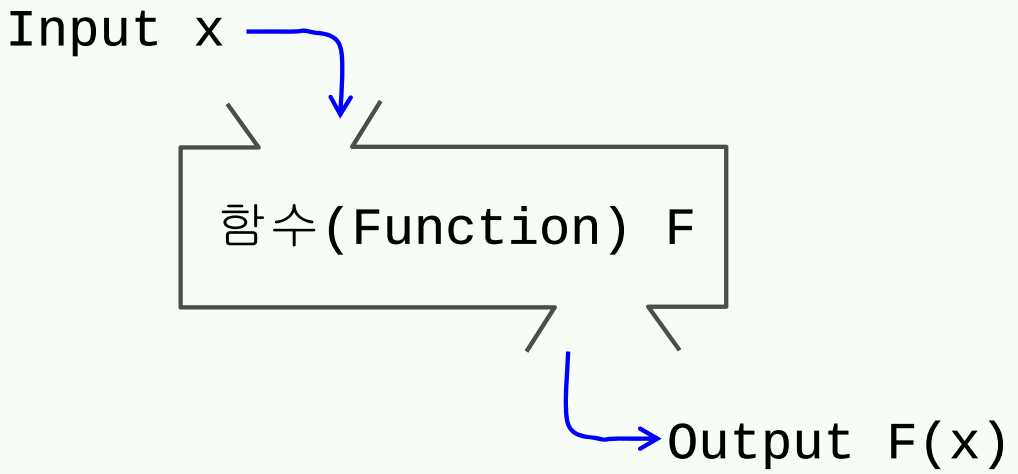
최회련

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

- **MySQL에서 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
- 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

변수 사용

- 변수 지정 예제 **(**전체를 선택한 후 실행시킴**)**

```
Set @myVar1 = 5;
Set @myVar2 = 3;
Set @myVar3 = 4.25;
Set @myVar4 = '고객이름 → ';

Select @myVar1;
Select @myVar2 + @myVar3;

Select @myVar4, name from customer where address = '서울';
```

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

```
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