# Day 26

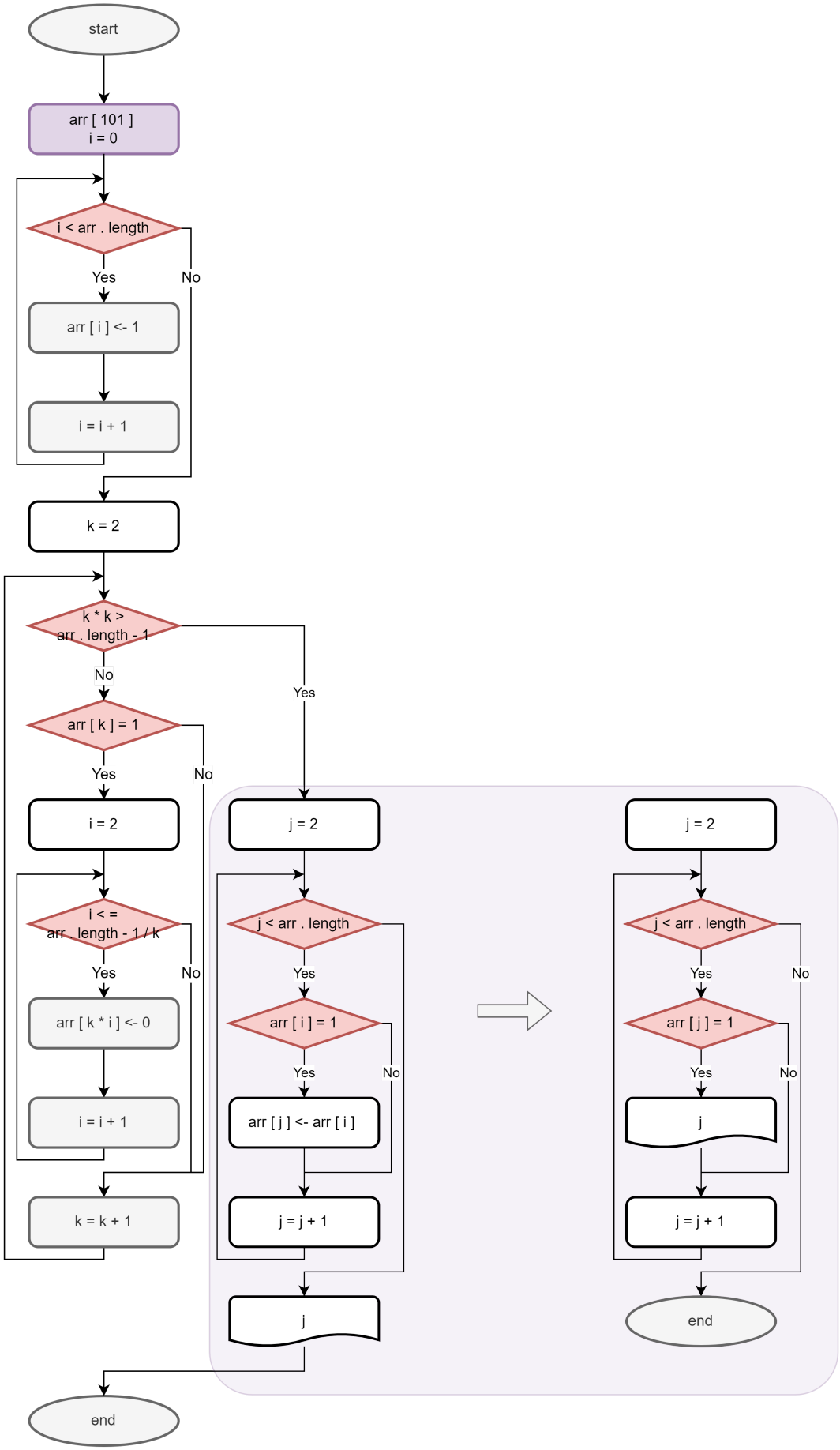
| | |
|---|---|
| Tags | |
| Date | @2022년 8월 4일 |
| 강의 번호 | |

-Algorithm-

< 에라토스테네스의 체 Sieve of Eratosthenes >

: flow chart에서 논리적 오류 → 불필요한 반복이 들어간다 ⇒ i = k 로 수정.

-. flow chart 완성과 java code 작성


start
arr [ 101 ]
i = 0
i < arr . length
Yes
No
arr [ i ] <- 1
i = i + 1
k = 2
k * k >
arr . length - 1
No
Yes
arr [ k ] = 1
Yes
No
i = 2
i < =
arr . length - 1 / k
Yes
No
arr [ k * i ] <- 0
i = i + 1
k = k + 1
j = 2
j < arr . length
Yes
arr [ i ] = 1
Yes
No
arr [ j ] <- arr [ i ]
j = j + 1
j
end
j = 2
j < arr . length
Yes
No
arr [ j ] = 1
Yes
No
j
j = j + 1
end

```
public class Eratos {

  public static void main(String[] args) {
    // 101의 배열을 만들어 1로 초기화

    int[] arr = new int[101];//0으로 초기화

    int k = 2;

    while(k * k < 100) {
      if (arr[k]==0) {
        int i = k;
          while( i <= 100 /k) {
            arr[k*i] = 1;
            i++;
          }
      }
      k++;
    }

    for (int j = 2 ; j < 100 ; j++) {
      if (arr[j] == 0) {
        System.out.print(j + " ");
      }
    }
  }

}
```

< 유클리드 알고리즘 Euclidean Algorithms >

-. 최대 공약수를 구하는 알고리즘

-. 최대 공약수는 약수들 중에서 가장 큰 수를 말한다.

: 반복 구조를 이용하는 중요한 알고리즘

=약수=

3의 약수 - 1, 3

4의 약수 - 1, 2, 4

5의 약수 - 1, 5

6의 약수 - 1, 2, 3, 6

8의 약수 - 1, 2, 4, 8

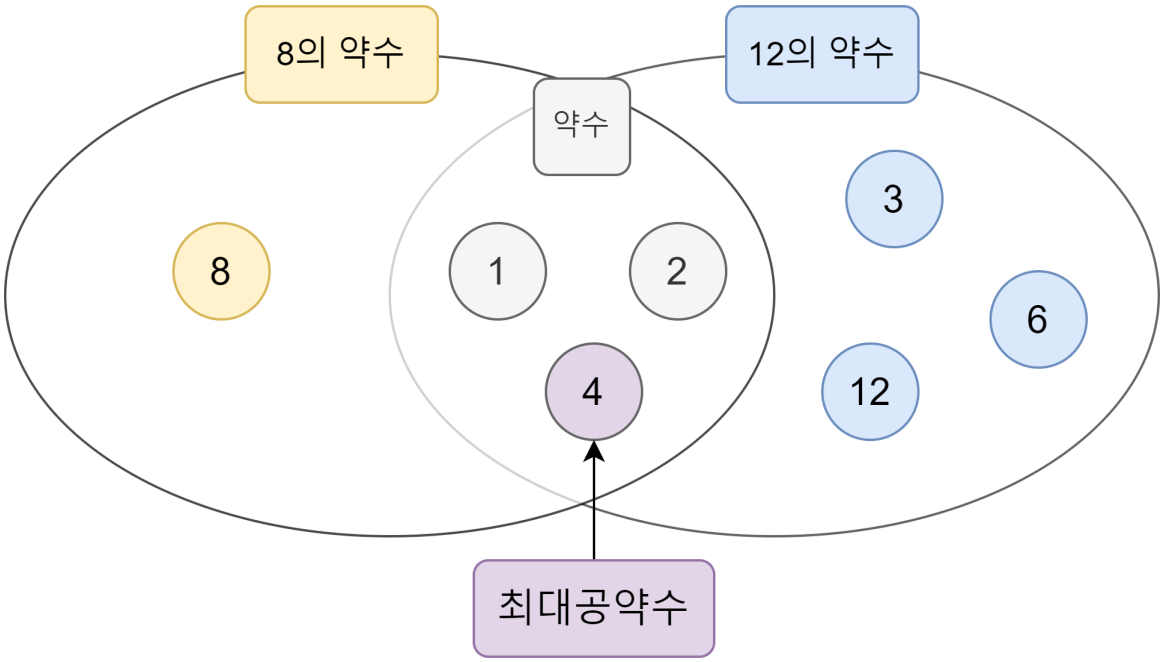
12의 약수 - 1, 2, 3, 4, 6, 12

=공약수=

8과 12의 공통된 약수 - 1, 2, 4

=최대 공약수=

두 수의 공약수 중 최대값, 8과 12의 최대 공약수는 4이다.




-. 최대 공약수를 구하는 절차

: 먼저, 어떤 복수의 수를 **소수의 곱셈** 형태로 분해하자. → 소인수 분해

8 = 1 * 2 * 2 * 2

12 = 1 * 2 * 2 * 3

이들 중 공통되는 소수를 서로 곱한 수가 바로 두 수의 최대 공약수 이다.

1 * 2 * 2 = 4

그러나 이런 절차를 반복하는 것은 상당히 복잡하다.

어떤 수를 소인수 분해하려면 먼저 그 수 이하의 소수들을 모두 구해야 한다.

그리고 그 소수 중 작은 숫자 부터 순서대로 원래의 수를 나누고, 나누어지지 않으면 그 다음 소수의 순서로 계속 계산을 반복해야 한다.

따라서, 단순해 보이지만 절차는 상당히 복잡해진다.

이러한 복잡성에 비해 매우 간단한 방법으로 최대 공약수를 구하는 것이 바로 ' 유클리드 알고리즘 ' 이다.

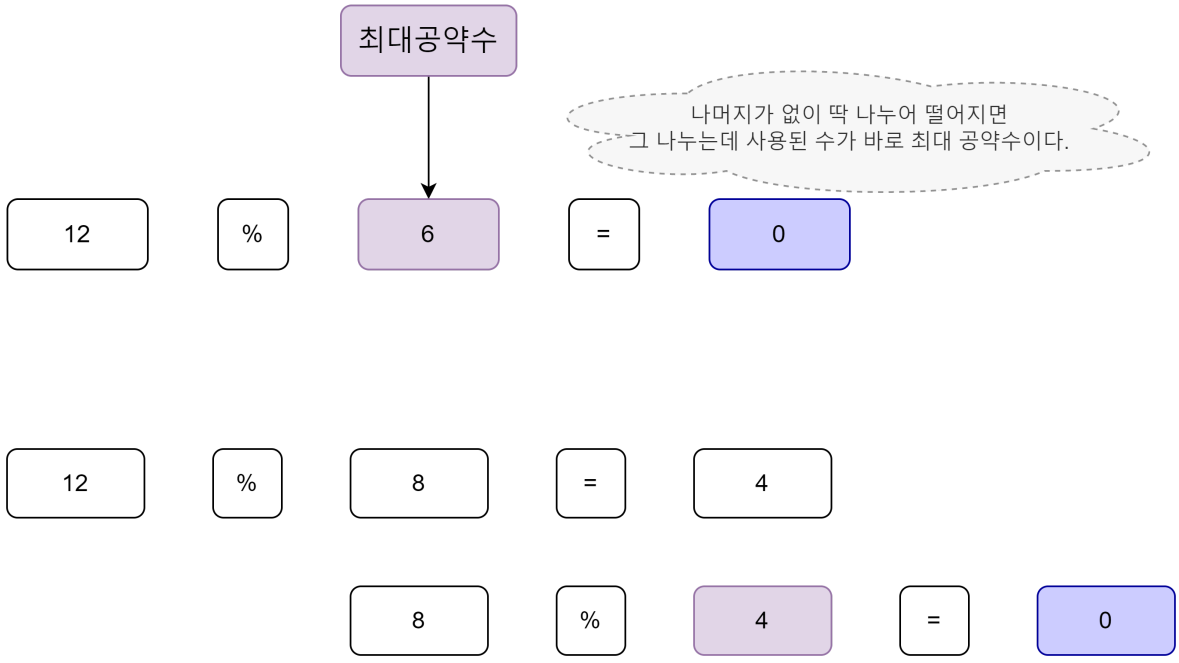
-. 유클리드 알고리즘 이란 ???

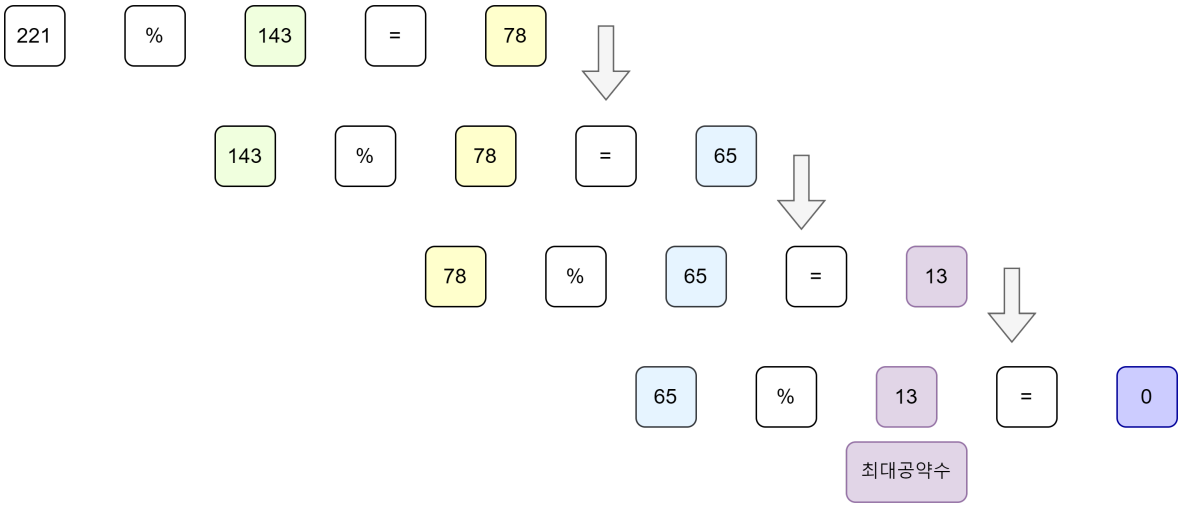
: 약 2,300년 전의 고대 그리스의 수학자로 수많은 수학적 이론을 생각해냈다. 그 중 하나가 바로 유클리드 알고리즘이다. 간단히 말하면 두 수의 나눗셈을 반복하여 최대 공약수를 수하는 것이다.

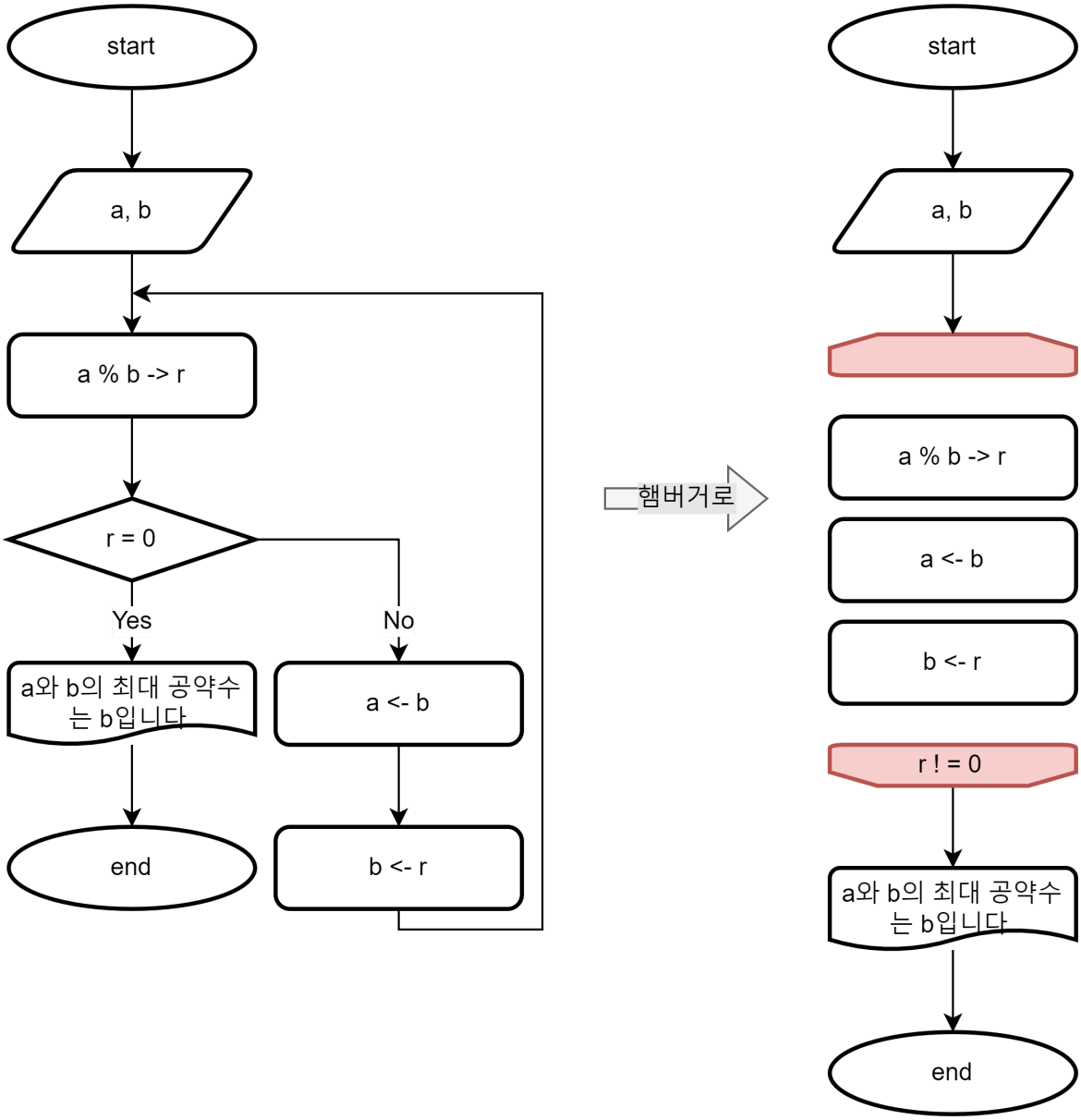
-. 그러면 어떻게 나눗셈을 반복할까...?

: 먼저, 큰 수를 작은 수로 나눈다. 나머지가 나오지 않게 되면 ( % ) → 그 때의 나누는데 사용된 작은 수가 바로 **최대공약수** 이다.

예를 들면, 12와 6의 최대 공약수는 6이다.




221
%
143
=
78
143
%
78
=
65
78
%
65
=
13
65
%
13
=
0
최대공약수


start
a, b
a % b -> r
r = 0
Yes
No
a와 b의 최대 공약수는 b입니다
a <- b
end
b <- r
햄버거로
start
a, b
a % b -> r
a <- b
b <- r
r ! = 0
a와 b의 최대 공약수는 b입니다
end

```java
class Main {
  public static void main(String[] args) {
    int a = 221;
    int b = 143;
    int r = 0;

    do{
      r = a % b;
      a = b;
      b = r;
    }while(r != 0);
    System.out.println(a);
  }
}
```

```
13
```

-DB-

-. chapter 6. ( p. 113 ~ )

## Working with String Data

When working with string data, you will be using one of the following character data types:

Working with String Data 문자열 데이터 작업

`CHAR`

Holds fixed-length, blank-padded strings. MySQL allows `CHAR` values up to 255 characters in length, Oracle Database permits up to 2,000 characters, and SQL Server allows up to 8,000 characters.

CHAR 고정 : 지정한 크기보다 문자열이 작으면 나머지 공간을 공백으로 채워준다.

`varchar`

Holds variable-length strings. MySQL permits up to 65,535 characters in a `varchar` column, Oracle Database (via the `varchar2` type) allows up to 4,000 characters, and SQL Server allows up to 8,000 characters.

varchar 가변 : 입력 길이에 따라 변한다

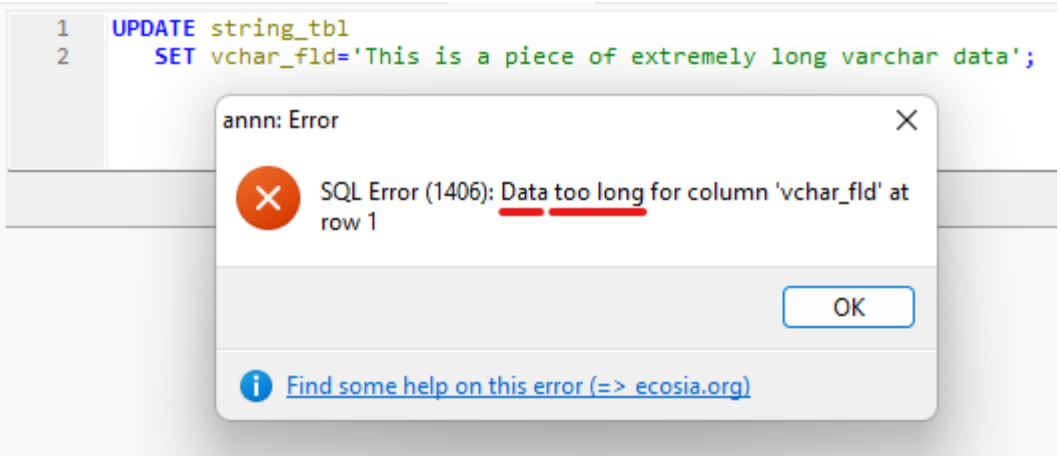
```
CREATE TABLE string_tbl
 (char_fld CHAR(30),
  vchar_fld VARCHAR(30),
  text_fld TEXT
 );
```

```
CREATE TABLE string_tbl
(char_fld CHAR(30),
vchar_fld VARCHAR(30),
text_fld TEXT
);
```

```
mysql> INSERT INTO string_tbl (char_fld, vchar_fld, text_fld)
    -> VALUES ('This is char data',
    ->   'This is varchar data',
    ->   'This is text data');
Query OK, 1 row affected (0.00 sec)
```

```
INSERT INTO string_tbl(char_fld,vchar_fld,text_fld)
VALUES ('This is char data',
'This is varchar data',
'This is text data');
```

```
mysql> UPDATE string_tbl
    -> SET vchar_fld = 'This is a piece of extremely long varchar data';
ERROR 1406 (22001): Data too long for column 'vchar_fld' at row 1
```




```
UPDATE string_tbl
SET vchar_fld='This is a piece of extremely long varchar data';
```

### Including single quotes

Since strings are demarcated by single quotes, you will need to be alert for strings that include single quotes or apostrophes. For example, you won't be able to insert the following string because the server will think that the apostrophe in the word *doesn't* marks the end of the string:

```
UPDATE string_tbl
SET text_fld = 'This string doesn't work';
```

Including single quotes 작은 따옴표를 포함하자.

To make the server ignore the apostrophe in the word *doesn't*, you will need to add an *escape* to the string so that the server treats the apostrophe like any other character in the string. All three servers allow you to escape a single quote by adding another single quote directly before, as in:

```
mysql> UPDATE string_tbl
    -> SET text_fld = 'This string didn''t work, but it does now';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

My SQL에서,

Oracle Database and MySQL users may also choose to escape a single quote by adding a backslash character immediately before, as in:

```
UPDATE string_tbl SET text_fld =
  'This string didn\'t work, but it does now'
```

오라클에서,

### Including special characters

If your application is multinational in scope, you might find yourself working with strings that include characters that do not appear on your keyboard. When working with the French and German languages, for example, you might need to include accented characters such as é and ö. The SQL Server and MySQL servers include the built-in function `char()` so that you can build strings from any of the 255 characters in the ASCII character set (Oracle Database users can use the `chr()` function). To demonstrate, the next example retrieves a typed string and its equivalent built via individual characters:

Including special characters 특수문자 포함하기

---

문자열 조작 String Manipulation ( p. 119 )

```
mysql> DELETE FROM string_tbl;
Query OK, 1 row affected (0.02 sec)
```

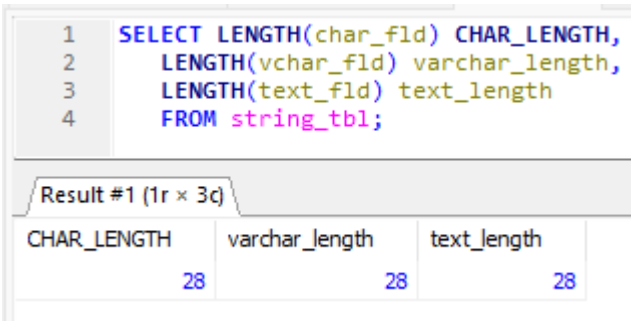
⇒ 테이블의 모든 데이터를 완전 삭제

```
DELETE FROM string_tbl;
```

```
mysql> INSERT INTO string_tbl (char_fld, vchar_fld, text_fld)
    -> VALUES ('This string is 28 characters',
    ->   'This string is 28 characters',
    ->   'This string is 28 characters');
Query OK, 1 row affected (0.00 sec)
```

```
INSERT INTO string_tbl(char_fld,vchar_fld,text_fld)
VALUES ('This string is 28 characters',
'This string is 28 characters',
'This string is 28 characters');
```

```
mysql> SELECT LENGTH(char_fld) char_length,
    ->   LENGTH(vchar_fld) varchar_length,
    ->   LENGTH(text_fld) text_length
    -> FROM string_tbl;
+-------------+----------------+-------------+
| char_length | varchar_length | text_length |
+-------------+----------------+-------------+
|          28 |             28 |          28 |
+-------------+----------------+-------------+
1 row in set (0.00 sec)
```




```
SELECT LENGTH(char_fld) CHAR_LENGTH,
LENGTH(vchar_fld) varchar_length,
LENGTH(text_fld) text_length
FROM string_tbl;
```

```
mysql> SELECT POSITION('characters' IN vchar_fld)
    -> FROM string_tbl;
+---------------------------------------+
| POSITION('characters' IN vchar_fld) |
+---------------------------------------+
|                                    19 |
+---------------------------------------+
1 row in set (0.12 sec)
```

```
mysql> SELECT LOCATE('is', vchar_fld, 5)
    -> FROM string_tbl;
+-----------------------------+
| LOCATE('is', vchar_fld, 5) |
+-----------------------------+
|                          13 |
+-----------------------------+
1 row in set (0.02 sec)
```

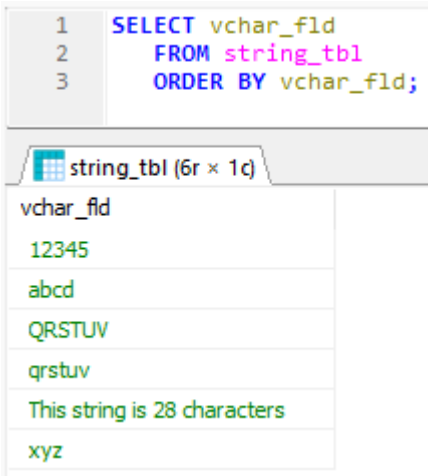
다섯번째 문자에서 시작하는 문자열 is의 위치를 찾는다.

Oracle Database does not include the `position()` or `locate()` function, but it does include the `instr()` function, which mimics the `position()` function when provided with two arguments and mimics the `locate()` function when provided with three arguments. SQL Server also doesn't include a `position()` or `locate()` function, but it does include the `charindx()` function, which also accepts either two or three arguments similar to Oracle's `instr()` function.

오라클에서는 position( )과 location( )를 사용할 수 없다.

```
mysql> INSERT INTO string_tbl(vchar_fld) VALUES ('abcd');
Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO string_tbl(vchar_fld) VALUES ('xyz');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO string_tbl(vchar_fld) VALUES ('QRSTUV');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO string_tbl(vchar_fld) VALUES ('qrstuv');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO string_tbl(vchar_fld) VALUES ('12345');
Query OK, 1 row affected (0.00 sec)
```

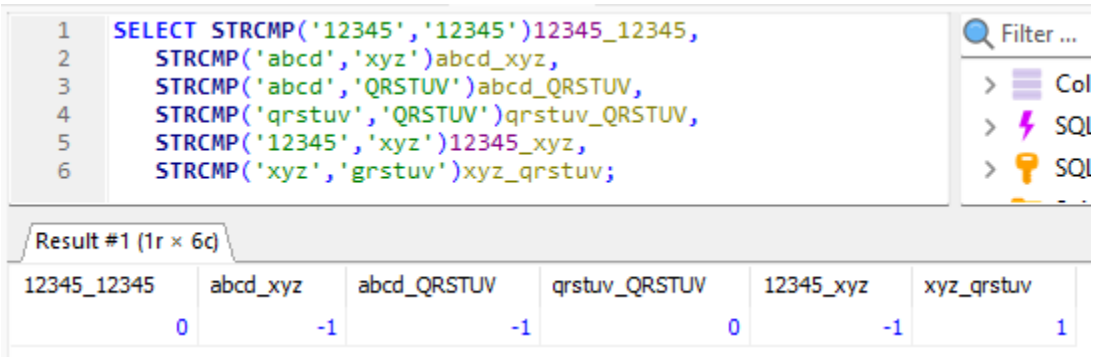
```
mysql> SELECT vchar_fld
    -> FROM string_tbl
    -> ORDER BY vchar_fld;
+-----------+
| vchar_fld |
+-----------+
| 12345     |
| abcd      |
| QRSTUV    |
| qrstuv    |
| xyz       |
+-----------+
5 rows in set (0.00 sec)
```




```
SELECT vchar_fld
FROM string_tbl
ORDER BY vchar_fld;
```

```
mysql> SELECT STRCMP('12345','12345') 12345_12345,
    ->   STRCMP('abcd','xyz') abcd_xyz,
    ->   STRCMP('abcd','QRSTUV') abcd_QRSTUV,
    ->   STRCMP('qrstuv','QRSTUV') qrstuv_QRSTUV,
    ->   STRCMP('12345','xyz') 12345_xyz,
    ->   STRCMP('xyz','qrstuv') xyz_qrstuv;
+-------------+----------+-------------+---------------+-----------+------------+
| 12345_12345 | abcd_xyz | abcd_QRSTUV | qrstuv_QRSTUV | 12345_xyz | xyz_qrstuv |
+-------------+----------+-------------+---------------+-----------+------------+
|           0 |       -1 |          -1 |             0 |        -1 |          1 |
+-------------+----------+-------------+---------------+-----------+------------+
1 row in set (0.00 sec)
```

'abcd'가 'xyz'작은게 앞에 오므로 -1
'xyz'이 'qrstuv'큰게 앞에 올 때는 1
대소문자는 구분하지 않으므로 'qrstuv'와 'QRSTUV'는 같아서 0




```
SELECT STRCMP('12345','12345')12345_12345,
STRCMP('abcd','xyz')abcd_xyz,
STRCMP('abcd','QRSTUV')abcd_QRSTUV,
STRCMP('qrstuv','QRSTUV')qrstuv_QRSTUV,
STRCMP('12345','xyz')12345_xyz,
STRCMP('xyz','grstuv')xyz_qrstuv;
```

```
mysql> SELECT name, name LIKE '%ns' ends_in_ns
    -> FROM department;
+----------------+------------+
| name           | ends_in_ns |
+----------------+------------+
| Operations     |          1 |
| Loans          |          1 |
| Administration |          0 |
+----------------+------------+
3 rows in set (0.25 sec)
```

'ns'로 끝나면 1을 반환하고 그렇지 않으면 0을 반환한다.

```
1  SELECT NAME,NAME LIKE '%ns' ends_in_ns
2     FROM department;
```

department (3r × 2c)

| NAME | ends_in_ns |
|---|---|
| Operations | 1 |
| Loans | 1 |
| Administration | 0 |

```
SELECT NAME,NAME LIKE '%ns' ends_in_ns
FROM department;
```

```
mysql> SELECT cust_id, cust_type_cd, fed_id,
    ->   fed_id REGEXP '.{3}-.{2}-.{4}' is_ss_no_format
    -> FROM customer;
```

---

### String functions that return strings

In some cases, you will need to modify existing strings, either by extracting part of the string or by adding additional text to the string. Every database server includes multiple functions to help with these tasks. Before I begin, I once again reset the data in the `string_tbl` table:

문자열을 반환하는 문자열 함수

```
mysql> DELETE FROM string_tbl;
Query OK, 5 rows affected (0.00 sec)

mysql> INSERT INTO string_tbl (text_fld)
    -> VALUES ('This string was 29 characters');
Query OK, 1 row affected (0.01 sec)
```

```
mysql> UPDATE string_tbl
    -> SET text_fld = CONCAT(text_fld, ', but now it is longer');
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

The contents of the text_fld column are now as follows:

```
mysql> SELECT text_fld
    -> FROM string_tbl;
+-----------------------------------------------------+
| text_fld                                            |
+-----------------------------------------------------+
| This string was 29 characters, but now it is longer |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT CONCAT(fname, ' ', lname, ' has been a ',
    ->   title, ' since ', start_date) emp_narrative
    -> FROM employee
    -> WHERE title = 'Teller' OR title = 'Head Teller';
```

문자열을 반환하는 CONCAT( ) 함수는 문자열이 저장된 데이터를 바꾸기 위한 용도로 사용될 수 있고
UPDATE - 영구변환
각 데이터 조각을 합쳐서 문자열을 만들어 임시로 보는 용도로도 사용된다. SELECT - 임시 변환

---

Working with Numeric Data 숫자로 데이터 작업하기 ( p. 126 )

Performing Arithmetic Functions 산술 함수 수행

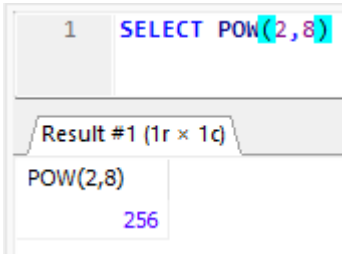
```
mysql> SELECT MOD(22.75, 5);
+---------------+
| MOD(22.75, 5) |
+---------------+
|          2.75 |
+---------------+
1 row in set (0.02 sec)
```

```
mysql> SELECT MOD(10,4);
+-----------+
| MOD(10,4) |
+-----------+
|         2 |
+-----------+
1 row in set (0.02 sec)
```

나머지 연산자

```
mysql> SELECT POW(2,8);
+----------+
| POW(2,8) |
+----------+
|      256 |
+----------+
1 row in set (0.03 sec)
```

2의 8승.




```
SELECT POW(2,8)
```

---

Controlling Number Precision 숫자 자리수 제어 ( p. 128 )

ceil(), floor(), round(), and truncate()

```
mysql> SELECT CEIL(72.445), FLOOR(72.445);
+--------------+---------------+
| CEIL(72.445) | FLOOR(72.445) |
+--------------+---------------+
|           73 |            72 |
+--------------+---------------+
1 row in set (0.06 sec)
```

= 올림 | 내림 =

```
mysql> SELECT ROUND(72.49999), ROUND(72.5), ROUND(72.50001);
+-----------------+-------------+-----------------+
| ROUND(72.49999) | ROUND(72.5) | ROUND(72.50001) |
+-----------------+-------------+-----------------+
|              72 |          73 |              73 |
+-----------------+-------------+-----------------+
1 row in set (0.00 sec)
```

= 반올림 =

round( ) 함수에서 두번째 인수 즉, 반올림의 위치값을 생략하면 정수 값을 기준으로 자동 반올림 처리한다.

```
mysql> SELECT ROUND(72.0909, 1), ROUND(72.0909, 2), ROUND(72.0909, 3);
+------------------+------------------+------------------+
| ROUND(72.0909, 1) | ROUND(72.0909, 2) | ROUND(72.0909, 3) |
+------------------+------------------+------------------+
|             72.1 |            72.09 |           72.091 |
+------------------+------------------+------------------+
1 row in set (0.00 sec)
```

소수점 반올림 할 자리수 지정.

round( ) 함수의 두번째 인수인 위치값이 **양수일 때**는 **소수점 아래의 ~로 반올림**하고

**음수일 때**는 **소수점 위의 ~에서 반올림**한다.

Both `truncate()` and `round()` also allow a *negative* value for the second argument, meaning that numbers to the *left* of the decimal place are truncated or rounded. This might seem like a strange thing to do at first, but there are valid applications. For example, you might sell a product that can be purchased only in units of 10. If a customer were to order 17 units, you could choose from one of the following methods to modify the customer's order quantity:

```
mysql> SELECT ROUND(17, -1), TRUNCATE(17, -1);
+--------------+----------------+
| ROUND(17, -1) | TRUNCATE(17, -1) |
+--------------+----------------+
|           20 |             10 |
+--------------+----------------+
1 row in set (0.00 sec)
```

---

Working with Temporal Data 시간 데이터 작업 ( p. 130, 131 )

While the computer age has exacerbated the issue, people have been dealing with time zone differences since the early days of naval exploration. To ensure a common point of reference for timekeeping, fifteenth-century navigators set their clocks to the time of day in Greenwich, England. This became known as *Greenwich Mean Time*, or GMT. All other time zones can be described by the number of hours' difference from GMT; for example, the time zone for the Eastern United States, known as *Eastern Standard Time*, can be described as GMT –5:00, or five hours earlier than GMT.

Today, we use a variation of GMT called *Coordinated Universal Time*, or UTC, which is based on an atomic clock (or, to be more precise, the average time of 200 atomic clocks in 50 locations worldwide, which is referred to as *Universal Time*). Both SQL Server and MySQL provide functions that will return the current UTC timestamp (`getutcdate()` for SQL Server and `utc_timestamp()` for MySQL).

협정 세계 표준시

*Table 7-2. Date format components*

| Component | Definition | Range |
|---|---|---|
| YYYY | Year, including century | 1000 to 9999 |
| MM | Month | 01 (January) to 12 (December) |
| DD | Day | 01 to 31 |
| HH | Hour | 00 to 23 |
| HHH | Hours (elapsed) | -838 to 838 |
| MI | Minute | 00 to 59 |
| SS | Second | 00 to 59 |

Month 이 MM 으로 먼저 지정되어 있기 때문에 Minute은 MI 로 사용한 다는 것을 알아두자. ( Minute 을 임의로 MM 으로 사용할 수 없는 이유. )

String-to-date conversions 문자열을 날짜로 변환

the cast() function:

```
mysql> SELECT CAST('2008-09-17 15:30:00' AS DATETIME);
+---------------------------------------+
| CAST('2008-09-17 15:30:00' AS DATETIME) |
+---------------------------------------+
| 2008-09-17 15:30:00                   |
+---------------------------------------+
1 row in set (0.00 sec)
```

문자로 입력된 날짜를 진짜 날짜로 바꿔주는 함수.

```
mysql> SELECT CAST('2008-09-17' AS DATE) date_field,
    ->   CAST('108:17:57' AS TIME) time_field;
+------------+------------+
| date_field | time_field |
+------------+------------+
| 2008-09-17 | 108:17:57  |
+------------+------------+
1 row in set (0.00 sec)
```

Functions for generating dates 날짜 생성 관련 함수 ( p. 135, 136 )

```
UPDATE individual
SET birth_date = STR_TO_DATE('September 17, 2008', '%M %d, %Y')
WHERE cust_id = 9999;
```

```
UPDATE individual
SET birth_date=STR_TO_DATE('Septemver 17, 2008','%M %d, %Y')
WHERE cust_id=9999;
```

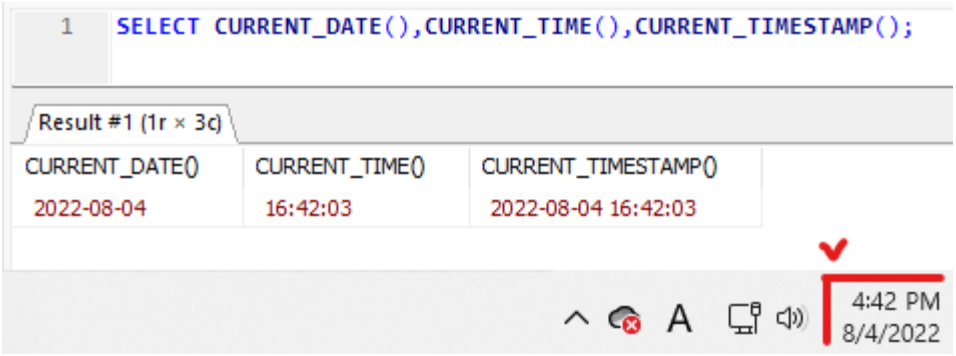
*Table 7-4. Date format components*

| Format component | Description |
|---|---|
| %M | Month name (January to December) |
| %m | Month numeric (01 to 12) |
| %d | Day numeric (01 to 31) |
| %j | Day of year (001 to 366) |
| %W | Weekday name (Sunday to Saturday) |
| %Y | Year, four-digit numeric |
| %y | Year, two-digit numeric |
| %H | Hour (00 to 23) |
| %h | Hour (01 to 12) |
| %i | Minutes (00 to 59) |
| %s | Seconds (00 to 59) |
| %f | Microseconds (000000 to 999999) |
| %p | A.M. or P.M. |

표현하는 방식이 차이가 있다.

```
mysql> SELECT CURRENT_DATE(), CURRENT_TIME(), CURRENT_TIMESTAMP();
+----------------+----------------+---------------------+
| CURRENT_DATE() | CURRENT_TIME() | CURRENT_TIMESTAMP() |
+----------------+----------------+---------------------+
| 2008-09-18     | 19:53:12       | 2008-09-18 19:53:12 |
+----------------+----------------+---------------------+
1 row in set (0.12 sec)
```

CURRENT_TIME 은 어느시간 일까 ? - **그 컴퓨터, 그 서버**의 시간이다.




```
SELECT CURRENT_DATE(),CURRENT_TIME(),CURRENT_TIMESTAMP();
```

```
mysql> SELECT DATE_ADD(CURRENT_DATE(), INTERVAL 5 DAY);
+-----------------------------------------+
| DATE_ADD(CURRENT_DATE(), INTERVAL 5 DAY) |
+-----------------------------------------+
| 2008-09-22                              |
+-----------------------------------------+
1 row in set (0.06 sec)
```

현재 날짜에서 5일을 더하는 방법

```
mysql> SELECT LAST_DAY('2008-09-17');
+------------------------+
| LAST_DAY('2008-09-17') |
+------------------------+
| 2008-09-30             |
+------------------------+
1 row in set (0.10 sec)
```

해당 달의 마지막 날 구하기
( 30 / 31 ??? )

Temporal functions that return strings 문자열로 반환하는 시간 함수 ( p. 139 )

```
mysql> SELECT DAYNAME('2008-09-18');
+-----------------------+
| DAYNAME('2008-09-18') |
+-----------------------+
| Thursday              |
+-----------------------+
1 row in set (0.08 sec)
```

요일로 반환.

```
mysql> SELECT EXTRACT(YEAR FROM '2008-09-18 22:19:05');
+-----------------------------------------+
| EXTRACT(YEAR FROM '2008-09-18 22:19:05') |
+-----------------------------------------+
|                                    2008 |
+-----------------------------------------+
1 row in set (0.00 sec)
```

어느 날짜에서 원하는 데이터를 추출. ( 년, 월, 일 등 )

Temporal Functions that return numbers 숫자를 반환하는 시간 함수 ( p. 140 )

```
mysql> SELECT DATEDIFF('2009-09-03', '2009-06-24');
+--------------------------------------+
| DATEDIFF('2009-09-03', '2009-06-24') |
+--------------------------------------+
|                                   71 |
+--------------------------------------+
1 row in set (0.05 sec)
```

두 날짜 사이의 기간을 계산 : 시간은 신경쓰지 않고 계산한다.

```
mysql> SELECT DATEDIFF('2009-06-24', '2009-09-03');
+--------------------------------------+
| DATEDIFF('2009-06-24', '2009-09-03') |
+--------------------------------------+
|                                  -71 |
+--------------------------------------+
1 row in set (0.01 sec)
```

앞 날짜가 과거면 - 값으로 나온다.

Conversion Functions 변환 함수

```
mysql> SELECT CAST('1456328' AS SIGNED INTEGER);
+-----------------------------------+
| CAST('1456328' AS SIGNED INTEGER) |
+-----------------------------------+
|                           1456328 |
+-----------------------------------+
1 row in set (0.01 sec)
```

형변환

cast( )를 사용하려면 값 또는 표현식 as 키워드.
변환 할 값의 자료형을 제공해야 한다.

```
mysql> SELECT CAST('999ABC111' AS UNSIGNED INTEGER);
+---------------------------------------+
| CAST('999ABC111' AS UNSIGNED INTEGER) |
+---------------------------------------+
|                                   999 |
+---------------------------------------+
1 row in set, 1 warning (0.08 sec)
```

문자열을 숫자로 변환할 때는 cast( )함수는 전체 문자열을 왼쪽에서 → 오른쪽으로 변환을 시도한다.
만약 숫자형 데이터가 아닌 문자가 있으면 **에러를 발생시키지는 않고 변환을 중지**한다.

```
mysql> show warnings;
+---------+------+-------------------------------------------------+
| Level   | Code | Message                                         |
+---------+------+-------------------------------------------------+
| Warning | 1292 | Truncated incorrect INTEGER value: '999ABC111' |
+---------+------+-------------------------------------------------+
1 row in set (0.07 sec)
```

이 경우 첫 세 글자들은 변환되지만 나머지는 버려진다. 모든 문자열이 변환되지는 않았다는 경고를 발생시킨다.

Grouping and Aggregates 그룹화와 집계 - chapter 8. ( p. 143 )

```
mysql> SELECT open_emp_id
    -> FROM account;
+-------------+
| open_emp_id |
+-------------+
|           1 |
|           1 |
|           1 |
|           1 |
|           1 |
|           1 |
|           1 |
|           1 |
|           1 |
|          10 |
|          10 |
|          10 |
|          10 |
|          10 |
|          10 |
|          10 |
|          10 |
|          13 |
```

```
1  SELECT open_emp_id
2     FROM account;
```

account (24r × 1c)

| open_emp_id |
|---|
| 1 |
| 1 |
| 1 |
| 1 |
| 1 |
| 1 |
| 1 |
| 1 |
| 10 |
| 10 |
| 10 |
| 10 |
| 10 |
| 10 |
| 10 |
| 13 |
| 13 |
| 13 |
| 16 |
| 16 |
| 16 |
| 16 |
| 16 |
| 16 |

```
SELECT open_emp_id
FROM account;
```

```
1  SELECT open_emp_id
2     FROM account
3     GROUP BY open_emp_id;
```

account (4r × 1c)

| open_emp_id |
|---|
| 1 |
| 10 |
| 13 |
| 16 |

```
mysql> SELECT open_emp_id
    -> FROM account
    -> GROUP BY open_emp_id;
+-------------+
| open_emp_id |
+-------------+
|           1 |
|          10 |
|          13 |
|          16 |
+-------------+
4 rows in set (0.00 sec)
```

```
SELECT open_emp_id
FROM account
GROUP BY open_emp_id;
```

```
mysql> SELECT open_emp_id, COUNT(*) how_many
    -> FROM account
    -> GROUP BY open_emp_id;
+-------------+----------+
| open_emp_id | how_many |
+-------------+----------+
|           1 |        8 |
|          10 |        7 |
|          13 |        3 |
|          16 |        6 |
+-------------+----------+
4 rows in set (0.00 sec)
```

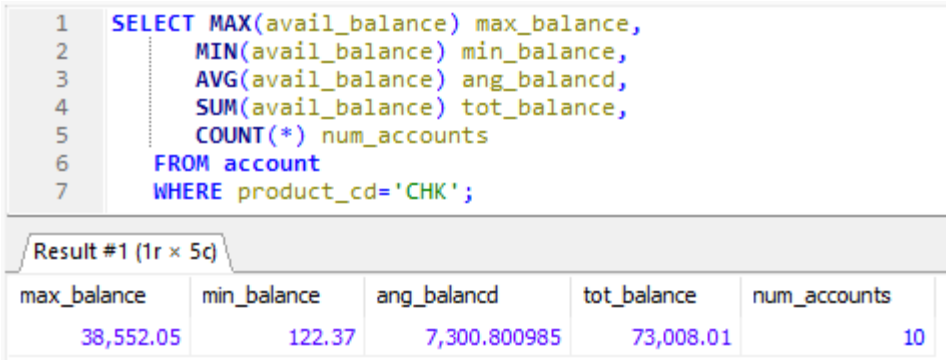
```
mysql> SELECT open_emp_id, COUNT(*) how_many
    -> FROM account
    -> WHERE COUNT(*) > 4
    -> GROUP BY open_emp_id;
ERROR 1111 (HY000): Invalid use of group function
```

Where 절이 적용 될 때 그룹이 아직 생성되지 않았기 때문에 where절에서 count( * ) 집계함수를 사용할 수 없다.

```
mysql> SELECT open_emp_id, COUNT(*) how_many
    -> FROM account
    -> GROUP BY open_emp_id
    -> HAVING COUNT(*) > 4;
+-------------+----------+
| open_emp_id | how_many |
+-------------+----------+
|           1 |        8 |
|          10 |        7 |
|          16 |        6 |
+-------------+----------+
3 rows in set (0.00 sec)
```

대신 그룹 필터 조건을 having 절에 넣어야 한다.

```
mysql> SELECT MAX(avail_balance) max_balance,
    ->   MIN(avail_balance) min_balance,
    ->   AVG(avail_balance) avg_balance,
    ->   SUM(avail_balance) tot_balance,
    ->   COUNT(*) num_accounts
    -> FROM account
    -> WHERE product_cd = 'CHK';
+-------------+-------------+-------------+-------------+--------------+
| max_balance | min_balance | avg_balance | tot_balance | num_accounts |
+-------------+-------------+-------------+-------------+--------------+
|    38552.05 |      122.37 | 7300.800985 |    73008.01 |           10 |
+-------------+-------------+-------------+-------------+--------------+
1 row in set (0.09 sec)
```




```
SELECT MAX(avail_balance) max_balance,
MIN(avail_balance) min_balance,
AVG(avail_balance) ang_balancd,
SUM(avail_balance) tot_balance,
COUNT(*) num_accounts
```

```
FROM account
WHERE product_cd='CHK';
```

The results from this query tell you that, across the 10 checking accounts in the account table, there is a maximum balance of $38,552.05, a minimum balance of $122.37, an average balance of $7,300.80, and a total balance across all 10 accounts of $73,008.01. Hopefully, this gives you an appreciation for the role of these aggregate functions; the next subsections further clarify how you can utilize these functions.

이 쿼리의 결과가 말하고자 하는 것은, account table에서 10개의 accounts(계정)를 거쳐서 확인했다.
최대 밸런스는 38,552.05달러, 최소 밸런스는 122.37달러, 평균 밸런스는 7,300.80달러이다.
그리고 토탈 밸런스(10개의 계정을 모두 거치는)는 73,008.01달러이다.