


Day 24

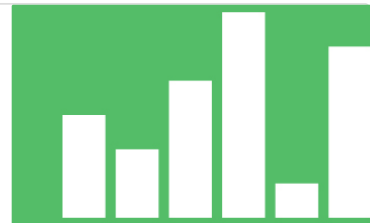
☰ Tags	Algorithm JAVA SQL flow chart
📅 Date	@2022년 8월 2일
▼ 강의 번호	AUS 101

-Algorithm-

Sorting (Bubble, Selection, Insertion, Merge, Quick, Counting, Radix) - VisuAlgo

Sorting is a very classic problem of reordering items (that can be compared, e.g., integers, floating-point numbers, strings, etc) of an array (or a list) in a certain order (increasing, non-decreasing (increasing or flat), decreasing, non-increasing (decreasing or flat),

 <https://visualgo.net/en/sorting>



알고리즘 진행을 그림으로 쉽게 이해할 수 있도록 만들어진 페이지.
(공부할 때 이용해 보자.)

< Quick Sort >

- . 가장 빠르다. 정렬의 최강자 끝.판.왕. !
- . 데이터를 대소그룹 둘로 나누어 분해한 후에 전체를 최종적으로 정렬하는 방식의 알고리즘이다.
- . Divide and conquer (분할 정복법)
- . 퀵정렬은 대량의 데이터를 정렬할 때 매우 자주 사용된다. 유명한 알고리즘 중에서도 실제로 많이 사용되는 빈도가 가장 높고 중요한 알고리즘이기도 한다.
- . 퀵정렬은 ' 기준값을 선택한 후 그 보다 작은 데이터 그룹과 큰 데이터 그룹으로 나눈다. ' 라는 처리를 반복 수행하여 데이터를 정렬하게 된다.

Quick Sort

5	4	7	6	8	3	1	2	9
0	1	2	3	4	5	6	7	8

5	4	7	6	8	3	1	2	9
0	1	2	3	4	5	6	7	8

맨 앞의 공을 기준으로 한다.

	4	7	6	8	3	1	2	9
0	1	2	3	4	5	6	7	8

기준보다 큰 공들은 기준 뒤로
작은 공들은 기준 앞으로 이동한다.

5

3	4	2	1	5	8	6	7	9
0	1	2	3	4	5	6	7	8

기준보다 큰 공들은 기준 뒤로
작은 공들은 기준 앞으로 이동한다.

3	4	2	1
0	1	2	3

8	6	7	9
5	6	7	8

맨 앞의 공을 기준으로 한다.

2	1	3	4
0	1	2	3

7	6	8	9
5	6	7	8

기준보다 큰 공들은 기준 뒤로
작은 공들은 기준 앞으로 이동한다.

2	1
0	1

7	6
5	6

맨 앞의 공을 기준으로 한다.

1	2
0	1

6	7
5	6

기준보다 큰 공들은 기준 뒤로
작은 공들은 기준 앞으로 이동한다.

1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8

[퀵 정렬의 알고리즘]

: 퀵 정렬은 크게 2개의 처리로 구성된다.

1. 기준값을 경계로 데이터를 대소로 나누는 처리

2. 나눈 데이터에 대해 반복적으로 똑같은 작업을 실행

- 1) 기준값을 경계로 데이터를 대소로 나누는 처리

-. 퀵정렬의 핵심은 데이터를 대소로 나누는 처리이다.

-. 배열의 왼쪽과 오른쪽부터 각각 변수를 움직여 대소로 정렬하자.

: 기준값보다 작은 공을 기준값의 앞으로 이동시키고 기준값보다 큰 공은 뒤로 이동시키는 것이 바로 퀵 정렬의 초석이 되는 처리이다.

-. 배열 설정 : 먼저 배열을 준비하자. 정수형 배열로 이름은 arr 로, 요소수는 9개로 정한다. 따라서, 첨자는 0 부터 8까지 사용된다.

5	4	7	6	8	3	1	2	9
0	1	2	3	4	5	6	7	8

arr 배열 설정.

-. 변수 설정 : 변수는 5개를 준비한다.

left - 정렬 범위에서 맨 앞 요소에 첨자를 넣는 변수

right - 정렬 범위에서 맨 끝 요소에 첨자를 넣는 변수

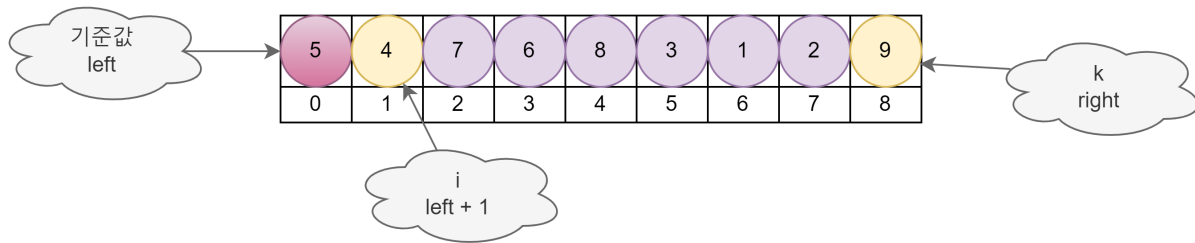
i - 기준값보다 큰 요소를 찾기 위한 변수

k - 기준값보다 작은 요소를 찾기 위한 변수

w - 데이터 교환용 임시 변수 temp

⇒ 이 다섯개의 변수를 사용하여 우선 left 와 right 에 각각 정렬 범위 맨 앞 요소의 첨자와 마지막 요소의 첨자를 대입한다. 따라서, 이번에는 (처음에는)left = 0, right = 8이 된다.

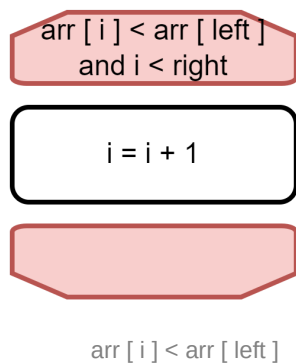
기준은 맨 앞 요소로 하기 때문에 arr [left] 이 된다. 그리고 i에 left의 하나 오른쪽인 left + 1 로 정하고 k 에는 right 을 대입한다.



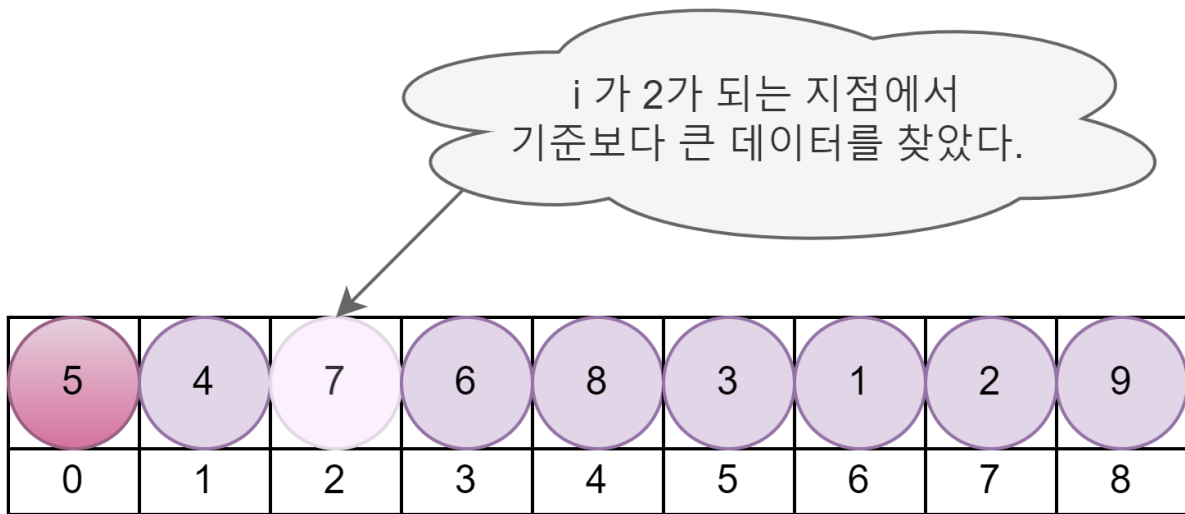
1. 변수 i를 사용하여 기준값보다 큰 요소 찾기

- i는 '기준값보다 큰 요소를 찾는 변수'이다. 현재 위치에서 하나씩 오른쪽으로 이동하면서 기준값보다 큰 요소가 있는지 확인하고 발견되면 그곳에서 멈춘다.

$arr[i] > arr[left]$



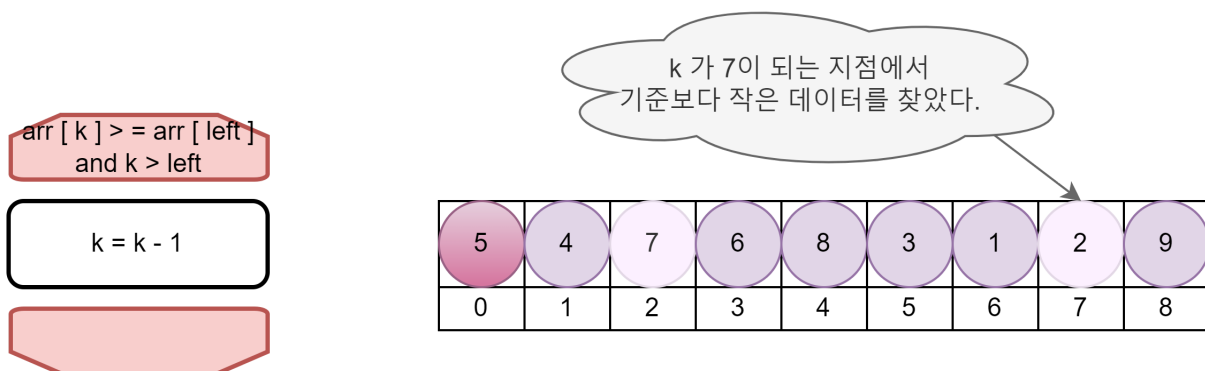
2가지 조건, 기준값 $arr[left]$ 보다 큰 값을 찾고 오른쪽 끝까지 찾지 못할 때 까지 반복



기준값보다 큰 요소를 발견했기 때문에 i는 일단 여기서 멈춘다. 그리고 반대쪽 변수 k 즉, 작은값 찾기로 넘어간다.

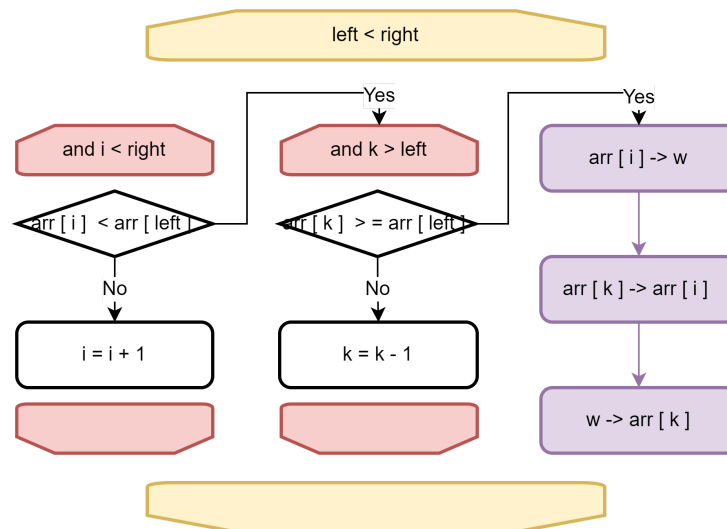
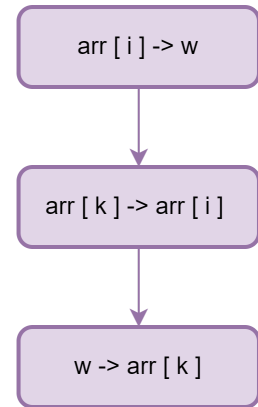
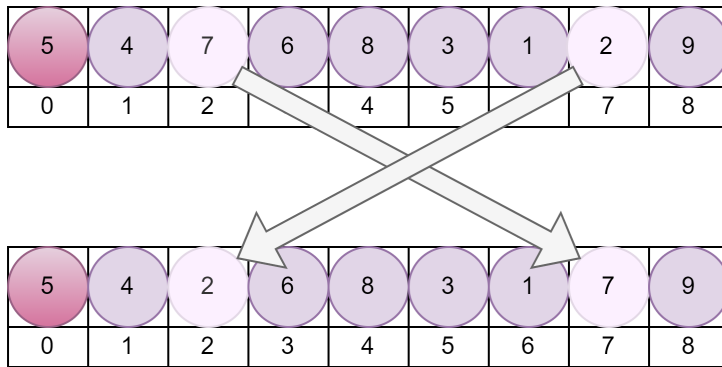
2. 변수 k를 사용하여 기준값보다 작은 요소 찾기

-. k는 ' 기준값보다 작은 요소를 찾는 변수 ' 이다. 현재 위치에서 하나씩 왼쪽으로 이동하면서 기준값보다 작은 요소가 있는지 확인하고 발견되면 그곳에서 멈춘다.



기준값보다 작은 요소를 발견했기 때문에 k도 일단 여기에서 멈춘다.

3. 큰 데이터와 작은 데이터 교환하기



```

import java.util.Arrays;

public class QuickSort {

    public static void main(String[] args) {
        int[] arr = {5,4,7,6,8,3,1,2,9};
        arr = quickSort(arr, 0, arr.length-1); //어레이, 시작, 끝
        System.out.println(Arrays.toString(arr));
    }

    static int[] quickSort(int[] arr, int start, int end) {
        int p = partition(arr, start, end);

        if(start < p-1)
            quickSort(arr, start, p-1);
        if(p<end)
            quickSort(arr, p, end);
        return arr;
    }
}
  
```

```

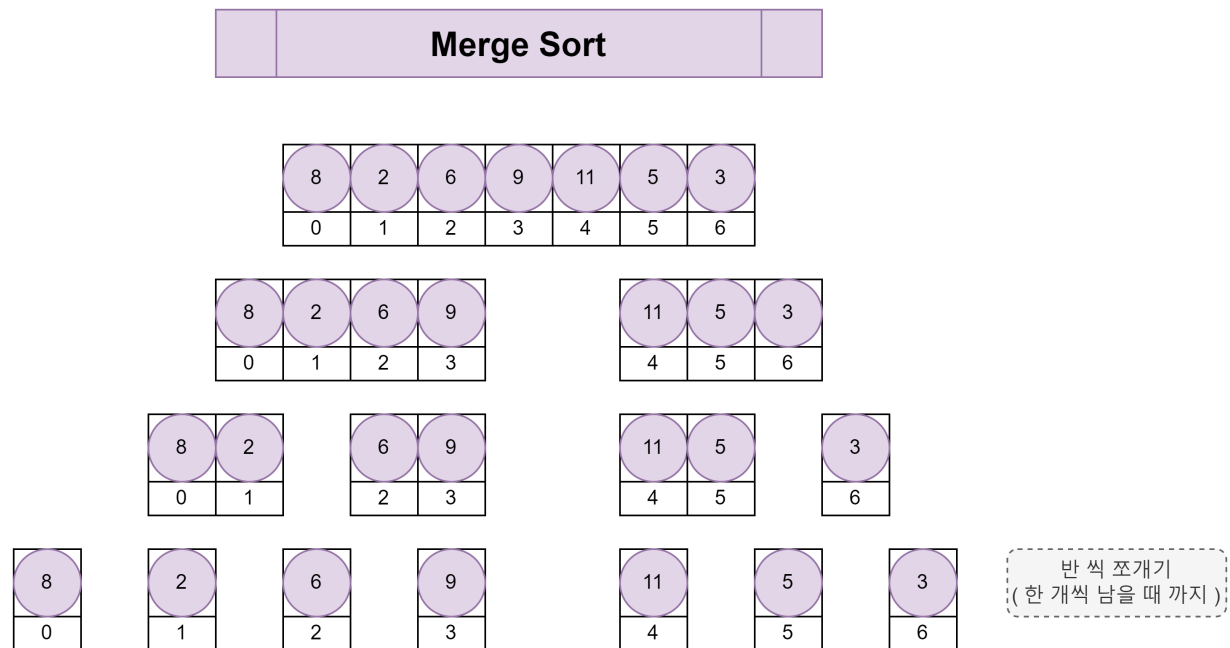
static int partition(int[] arr, int start, int end) {
    int pivot = arr[(start+end)/2];
    while(start <= end) {
        while(arr[start]<pivot) start++;
        while(arr[end]>pivot) end--;
        if(start <= end) {
            int tmp = arr[start];
            arr[start] = arr[end];
            arr[end] = tmp;
            start++;
            end--;
        }
    }
    return start;
}
}

```

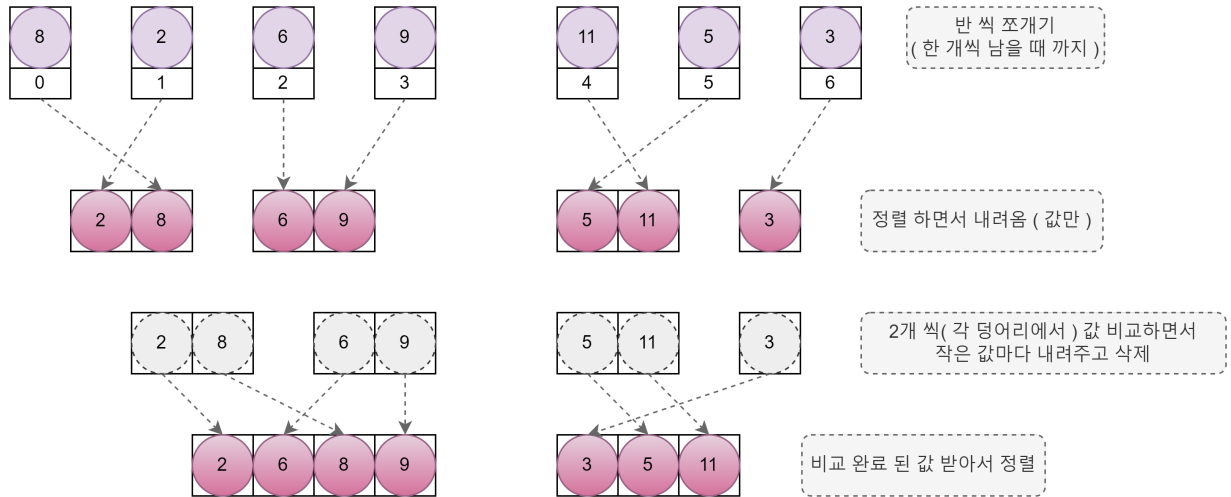
< Merge Sort > : 병합 정렬

- Quick Sort 다음으로 가장 빠르다.

- 분할 정복 (Divide and Conquer)을 사용하는 방법이다. 분할 정복은 주어진 문제를 해결하기 쉬운 단계 까지 분할 한 후에 분할 된 문제를 해결하고 그 결과를 다시 결합하는 알고리즘이다.

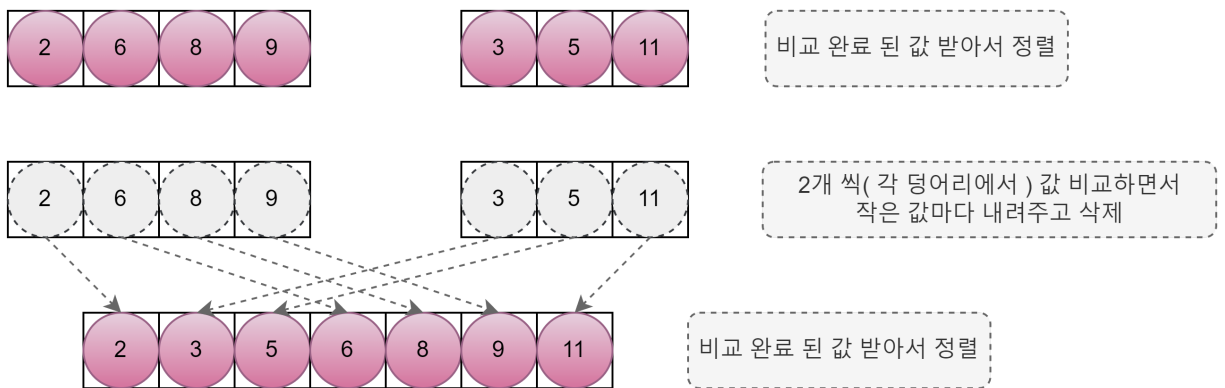


먼저 데이터들을 정렬을 생각하지 않고 1개씩 될 때까지 나눈다. 나누는 방법은 배열을 반으로 쪼개고 그 쪼개진 배열을 또 다시 반으로 쪼개고 이 과정을 반복한다. 위에서 7개의 데이터를 분할하기 위해 이 과정을 3번 거쳤다.



분할이 완료된 후에는 데이터를 비교하면서 결합한다. 제일 앞에있는 2와 8을 다시 합치는데 작은 수 2가 앞으로, 큰 수 8이 뒤로 보낸 상태로 결합한다.

이 과정을 각각 2개씩 반복하여 합치고 그 합친 2개를 다시 합칩니다.



이 때 각각의 그룹에서 먼저 제일 첫번째 값을 비교하여 작은 값을 추출한다. 그 다음 다시 한번 각각의 그룹의 제일 왼쪽 즉, 작은 값을 또 다시 비교하여 둘 중 작은 값을 다시 추출한다. 이 과정을 반복하여 전체 정렬을 마치게 된다.

-DATABASE (SQL)-

=복습=

-. 데이터 베이스 → 아주 중요하다.

-. 문자데이터, 텍스트 (긴 텍스트) : varchar, 숫자 : int, 날짜 : timestamp 를 많이 쓴다. (날짜 - datetime 은 생년월일 입력받을 때 주로 씀.)

- Chapter 4 . 필터링

: 새로운 데이터 웨어하우스 피드를 준비할 때 사용한 테이블에서 모든 데이터 제거

: 새 열이 추가된 후 테이블의 **모든 행 수정**

: 메세지 큐 테이블에서 **모든 행 검색**

< 조건 평가 >

```
WHERE title = 'Teller' AND start_date < '2007-01-01'
```

두 가지 조건을 모두 and 충족하는 타이틀이 텔러이고, 동시에 날짜가 2007년 1월 1일 이전인 행만 결과 포함된다. 조건이 여러개 있어도 AND 연산자로 구분될 경우에는 결과셋에 모든 조건이 True 인 경우만 포함된다. 즉 true로 평가된다.

< 괄호 사용 >

Table 4-2. Three-condition evaluation using and, or

Intermediate result	Final result
WHERE true AND (true OR true)	True
WHERE true AND (true OR false)	True
WHERE true AND (false OR true)	True
WHERE true AND (false OR false)	False

세가지 조건이 있을 경우 최종 결과는 괄호안의 2가지 조건의 결과와 최종 마지막 결과의 평가에 따라 최종 결과가 정해지게 된다.

< not 연산자 사용 >

Table 4-3. Three-condition evaluation using and, or, and not

Intermediate result	Final result
WHERE true AND NOT (true OR true)	False
WHERE true AND NOT (true OR false)	False
WHERE true AND NOT (false OR true)	False
WHERE true AND NOT (false OR false)	True
WHERE false AND NOT (true OR true)	False
WHERE false AND NOT (true OR false)	False
WHERE false AND NOT (false OR true)	False
WHERE false AND NOT (false OR false)	False

not 연산자를 사용하여 평가 결과를 반대로. true의 경우는 false로, false의 경우에는 true. 결과를 뒤집을 수 있다.

AND 연산자 진리 연산표

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR 연산자 진리 연산표

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT 연산자 진리 연산표

NOT	TRUE	FALSE	NULL
TRUE	FALSE	TRUE	NULL

< 조건 작성 >

: 표현식은 아래의 내용들로 구성할 수 있다.

- 1) 숫자
- 2) 테이블 또는 뷰의 칼럼
- 3) 텔러와 같은 문자열
- 4) concat과 같은 내장 함수들
- 5) 서브쿼리, 헤드텔러 등등과 같은 표현식 목록

< 조건의 유형 >

동등조건 ‘ 열 = 값 ‘

부등조건 두 표현식이 동일하지 않을 때 사용 < >

< 동등조건을 사용한 데이터 수정 >

```
DELETE FROM account
WHERE status = 'CLOSED' AND YEAR(close_date) = 2002;
```

< 범위 조건 >

```
mysql> SELECT emp_id, fname, lname, start_date
-> FROM employee
-> WHERE start_date < '2007-01-01';
```

특정 범위내에 원하는 조건이 있는지를 확인하는 범위 조건을 작성할 수 있다.

이 유형은 보통 숫자 또는 시간데이터로 작업할 때 주로 발생한다.

Host: 127.0.0.1 Database: bank Table: employee Data Query*

```

1 SELECT emp_id, fname, lname, start_date
2 FROM employee
3 WHERE start_date < '2007-01-01';
4

```

employee (18r x 4c)

emp_id	fname	lname	start_date
1	Michael	Smith	2001-06-22
2	Susan	Barker	2002-09-12
3	Robert	Tyler	2000-02-09
4	Susan	Hawthorne	2002-04-24
5	John	Gooding	2003-11-14
6	Helen	Fleming	2004-03-17
7	Chris	Tucker	2004-09-15
8	Sarah	Parker	2002-12-02

```

SELECT emp_id, fname, lname, start_date
FROM employee
WHERE start_date < '2007-01-01';

```

< between 연산자 >

범위의 상한과 하한이 모두 있을 때 사용한 between 연산자를 사용하여 하나의 조건으로 사용할 수 있다.

```

mysql> SELECT emp_id, fname, lname, start_date
-> FROM employee
-> WHERE start_date BETWEEN '2005-01-01' AND '2007-01-01';

```

Host: 127.0.0.1 Database: bank Table: employee Data Query*

```

1 SELECT emp_id, fname, lname, start_date
2 FROM employee
3 WHERE start_date BETWEEN '2005-01-01' AND '2007-01-01';
4

```

employee (0r x 4c)

emp_id	fname	lname	start_date
--------	-------	-------	------------

empty set.

```

SELECT emp_id, fname, lname, start_date
FROM employee
WHERE start_date BETWEEN '2005-01-01' AND '2007-01-01';

```

```
mysql> SELECT account_id, product_cd, cust_id, avail_balance
-> FROM account
-> WHERE avail_balance BETWEEN 3000 AND 5000;
```

```
1 SELECT account_id, product_cd, cust_id, avail_balance
2 FROM account
3 WHERE avail_balance BETWEEN 3000 AND 5000;
```

account_id	product_cd	cust_id	avail_balance
3	CD	1	3,000.0
17	CD	7	5,000.0
18	CHK	8	3,487.19

```
SELECT account_id, product_cd, cust_id, avail_balance
FROM account
WHERE avail_balance BETWEEN 3000 AND 5000;
```

< 와일드 카드 사용하기 >

- **특정 문자**로 시작 or 종료하는 문자열.
- **부분 문자열**로 시작 or 종료하는 문자열.
- 문자열 내에 **특정 문자**를 포함하는 모든 문자열
- 문자열 내에 **특정 문자열**을 포함하는 모든 문자열
- 개별 문자에 관계 없이 특정한 형식의 문자열

Table 4-4. Wildcard characters

Wildcard character	Matches
_	Exactly one character
%	Any number of characters (including 0)

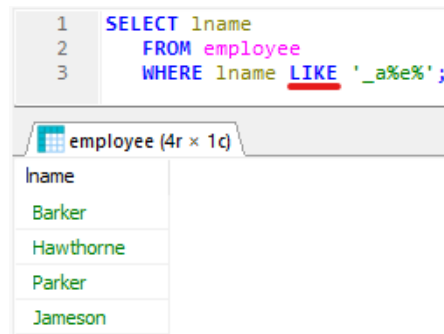
_ : 정확히 한 글자.

% : 0을 포함한 개수에 상관 없이 모든 문자

```
mysql> SELECT lname
-> FROM employee
-> WHERE lname LIKE '_a%e%';
```

lname
Barker
Hawthorne
Parker
Jameson

4 rows in set (0.00 sec)



```
SELECT lname
FROM employee
WHERE lname LIKE '_a%e%';
```

Table 4-5. Sample search expressions

Search expression	Interpretation
F%	Strings beginning with <i>F</i>
%t	Strings ending with <i>t</i>
%bas%	Strings containing the substring 'bas'
_ _t_	Four-character strings with a <i>t</i> in the third position
_ _ _ - _ - _ _	11-character strings with dashes in the fourth and seventh positions

```
mysql> SELECT cust_id, fed_id
-> FROM customer
-> WHERE fed_id LIKE '___-__-___';
```

cust_id	fed_id
1	111-11-1111
2	222-22-2222
3	333-33-3333
4	444-44-4444
5	555-55-5555

```
1 SELECT cust_id, fed_id
2 FROM customer
3 WHERE fed_id LIKE '___-__-___';
```

customer (9r x 2c)

cust_id	fed_id
1	111-11-1111
2	222-22-2222
3	333-33-3333
4	444-44-4444
5	555-55-5555
6	666-66-6666
7	777-77-7777
8	888-88-8888
9	999-99-9999

```
SELECT cust_id, fed_id
FROM customer
WHERE fed_id LIKE '--__';
```

```
mysql> SELECT emp_id, fname, lname
-> FROM employee
-> WHERE lname LIKE 'F%' OR lname LIKE 'G%';
```

emp_id	fname	lname
5	John	Gooding
6	Helen	Fleming
9	Jane	Grossman
17	Beth	Fowler

4 rows in set (0.00 sec)

1	SELECT emp_id, fname, lname
2	FROM employee
3	WHERE lname LIKE 'F%' OR lname LIKE 'G%';

employee (4r × 3c)		
emp_id	fname	lname
5	John	Gooding
6	Helen	Fleming
9	Jane	Grossman
17	Beth	Fowler

```
SELECT emp_id, fname, lname
FROM employee
WHERE lname LIKE 'F%' OR lname LIKE 'G%';
```

+) 정규표현식 : 표현하는 방식. 익혀두면 어디서든 그대로 사용가능하고 적용가능 한 언어이다. 코드를 효율적으로 이용할 수 있도록 줄여준다. Python에서 주로 적용가능하다. 안해도 상관없지만 여유있을 때 공부해두면 사용할 수 있어서 좋다.

< null >

- 해당사항 없음 (Not applicable)
- 아직 알려지지 않은 값 (Value not yet known)
- 정의 되지 않은 값 (Value undefined)

* 주의사항 *

When working with null, you should remember: Null일 수는 있지만 Null과 같을 수는 없다.

- An expression can *be* null, but it can never *equal* null.
- Two nulls are never equal to each other. 두개의 null 서로 같지 않다.

: null 일 수는 있지만 null 과 같을 수는 없다.

⇒ 뭔가가 null이니? 했을 때 null이라고 한다면 true가 안나옴. 비교를 하면 null일 때 true가 안나옴. 즉, null은 비교할 수 없는 존재라서. null은 어떤 값이 아니고 비어있는 것이기 때문에. 그래서 null 일 수는 있지만 null 과 같을 수는 없다는 것이다.

: 두개의 null은 서로 같지 않다.


```
mysql> SELECT emp_id, fname, lname, superior_emp_id
-> FROM employee
-> WHERE superior_emp_id IS NULL;
+-----+-----+-----+-----+
| emp_id | fname  | lname | superior_emp_id |
+-----+-----+-----+-----+
|      1 | Michael | Smith |                NULL |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

1	SELECT emp_id, fname, lname, superior_emp_id
2	FROM employee
3	WHERE superior_emp_id <u>IS NULL;</u>

employee (1r x 4c)			
emp_id	fname	lname	superior_emp_id
1	Michael	Smith	(NULL)

```
SELECT emp_id, fname, lname, superior_emp_id
FROM employee
WHERE superior_emp_id IS NULL;
```

```
mysql> SELECT emp_id, fname, lname, superior_emp_id
-> FROM employee
-> WHERE superior_emp_id = NULL;
Empty set (0.01 sec)
```

1	SELECT emp_id, fname, lname, superior_emp_id
2	FROM employee
3	WHERE superior_emp_id = NULL;

employee (0r x 4c)			
emp_id	fname	lname	superior_emp_id

Empty set.

```
SELECT emp_id, fname, lname, superior_emp_id
FROM employee
WHERE superior_emp_id = NULL;
```

```
mysql> SELECT emp_id, fname, lname, superior_emp_id
-> FROM employee
-> WHERE superior_emp_id IS NOT NULL;
```

1	SELECT emp_id, fname, lname, superior_emp_id
2	FROM employee
3	WHERE superior_emp_id IS NOT NULL;

employee (17r × 4c)				
emp_id	fname	lname	superior_emp_id	
2	Susan	Barker	1	
3	Robert	Tyler	1	
4	Susan	Hawthorne	3	
5	John	Gooding	4	
6	Helen	Fleming	4	
7	Chris	Tucker	6	
8	Sarah	Parker	6	
9	Jane	Grossman	6	
10	Paula	Roberts	4	
11	Thomas	Ziegler	10	
12	Samantha	Jameson	10	
13	John	Blake	4	
14	Cindy	Mason	13	
15	Frank	Portman	13	
16	Theresa	Markham	4	
17	Beth	Fowler	16	
18	Rick	Tulman	16	

```
SELECT emp_id, fname, lname, superior_emp_id
FROM employee
WHERE superior_emp_id IS NOT NULL;
```

=문제=

월급이 3000인 직원들의 이름, 월급, 직업을 출력해 보겠습니다.

Quiz 7.

ENAME	SAL	JOB
FORD	3000	ANALYST
SCOTT	3000	ANALYST

1	SELECT	ename,sal,job
2	FROM	emp
3	WHERE	sal=3000;

emp (2r × 3c)		
ename	sal	job
FORD	3,000	ANALYST
SCOTT	3,000	ANALYST

```
SELECT ename,sal,job
FROM emp
WHERE sal=3000;
```

월급이 3000 이상인 직원들의 이름과 월급을 출력;

Quiz 7_1.

이름	월급
KING	5000
FORD	3000
SCOTT	3000

1	SELECT	ename 이름 ,sal 월급
2	FROM	emp
3	WHERE	sal>=3000;

emp (3r × 2c)	
이름	월급
KING	5,000
FORD	3,000
SCOTT	3,000

```
SELECT ename 이름,sal 월급
FROM emp
WHERE sal>=3000;
```

이름이 SCOTT인 사원의 이름, 월급, 직업, 입사일, 부서 번호를 출력해 보겠습니다.

Quiz 8.

ENAME	SAL	JOB	HIREDATE	DEPTNO
SCOTT	3000	ANALYST	82/12/22	20

1	SELECT	ename,sal,job,hiredate,deptno
2	FROM	emp
3	WHERE	ename='SCOTT';

emp (1r × 5c)				
ename	sal	job	hiredate	deptno
SCOTT	3,000	ANALYST	1982-12-22	20

```
SELECT ename,sal,job,hiredate,deptno
FROM emp
WHERE ename='SCOTT';
```

연봉이 36000 이상인 사원들의 이름과 연봉을 출력해 보겠습니다.

Quiz 9.

ENAME	연봉
KING	60000
FORD	36000
SCOTT	36000

1	SELECT	ename,sal*12 연봉
2	FROM	emp
3	WHERE	sal*12>=36000;

emp (3r × 2c)	
ename	연봉
KING	60,000
FORD	36,000
SCOTT	36,000

```
SELECT ename,sal*12 연봉
FROM emp
WHERE sal*12>=36000;
```

월급이 1000에서 3000 사이인 직원들의 이름과 월급을 출력해 보겠습니다.

Quiz 10.

ENAME	SAL
BLAKE	2850
CLARK	2450
:	:
ADAMS	1100
MILLER	1300

1	SELECT	ename,sal
2	FROM	emp
3	WHERE	sal BETWEEN 1000 AND 3000;

emp (11r × 2c)	
ename	sal
BLAKE	2,850
CLARK	2,450
JONES	2,975
MARTIN	1,250
ALLEN	1,600
TURNER	1,500
WARD	1,250
FORD	3,000
SCOTT	3,000
ADAMS	1,100
MILLER	1,300

```
SELECT ename,sal
FROM emp
WHERE sal BETWEEN 1000 AND 3000;
```

1982년도에 입사한 직원들의 이름과 입사일을 조회:

Quiz 11.

ENAME	HIREDATE
SCOTT	82/12/22
MILLER	82/01/11

1	SELECT	ename,hiredate
2	FROM	emp
3	WHERE	hiredate BETWEEN '1982-01-01'AND'1982-12-31';

emp (2r × 2c)	
ename	hiredate
SCOTT	1982-12-22
MILLER	1982-01-11

```
SELECT ename,hiredate
FROM emp
WHERE hiredate BETWEEN '1982-01-01'AND'1982-12-31';
```

같은 결과 다른 방법 ▼

1	SELECT ename,hiredate
2	FROM emp
3	WHERE hiredate LIKE'1982-__-__';

emp (2r × 2c)	
ename	hiredate
SCOTT	1982-12-22
MILLER	1982-01-11

```
SELECT ename,hiredate
FROM emp
WHERE hiredate LIKE'1982--';
```

이름의 두 번째 철자가 M인 사원의 이름을 출력하

Quiz 12.

ENAME
SMITH

1	SELECT ename
2	FROM emp
3	WHERE ename LIKE' _M%';

emp (1r × 1c)	
ename	
SMITH	

```
SELECT ename
FROM emp
WHERE ename LIKE' _M%';
```

이름이 A가 포함된 직원들을 전부 검색:

Quiz 12_1.

1	SELECT	ename
2	FROM	emp
3	WHERE	ename LIKE '%A%';

emp (7r x 1c)	
ename	
BLAKE	
CLARK	
MARTIN	
ALLEN	
JAMES	
WARD	
ADAMS	

```
SELECT ename
FROM emp
WHERE ename LIKE '%A%';
```

커미션이 NULL인 직원들의 이름과 커미션을 출력해 보겠습니다.

Quiz 13.

ENAME	COMM
KING	
:	:
MILLER	

1	SELECT	ename,comm
2	FROM	emp
3	WHERE	comm IS NULL;

emp (10r × 2c)	
ename	comm
KING	(NULL)
BLAKE	(NULL)
CLARK	(NULL)
JONES	(NULL)
JAMES	(NULL)
FORD	(NULL)
SMITH	(NULL)
SCOTT	(NULL)
ADAMS	(NULL)
MILLER	(NULL)

```
SELECT ename,comm
FROM emp
WHERE comm IS NULL;
```

직업이 SALESMAN, ANALYST, MANAGER인 직원들의 이름, 월급, 직업을 출력해 보겠습니다.

Quiz 14.

ENAME	SAL	JOB
BLAKE	2850	MANAGER
CLARK	2450	MANAGER
JONES	2975	MANAGER
MARTIN	1250	SALESMAN
ALLEN	1600	SALESMAN
TURNER	1500	SALESMAN
WARD	1250	SALESMAN
FORD	3000	ANALYST
SCOTT	3000	ANALYST

1	SELECT	ENAME,SAL,JOB
2	FROM	emp
3	WHERE	job='SALESMAN'OR job='ANALYST'OR job='MANAGER';

emp (9r × 3c)		
ENAME	SAL	JOB
BLAKE	2,850	MANAGER
CLARK	2,450	MANAGER
JONES	2,975	MANAGER
MARTIN	1,250	SALESMAN
ALLEN	1,600	SALESMAN
TURNER	1,500	SALESMAN
WARD	1,250	SALESMAN
FORD	3,000	ANALYST
SCOTT	3,000	ANALYST

```

SELECT ENAME, SAL, JOB
FROM emp
WHERE job='SALESMAN'OR job='ANALYST'OR job='MANAGER';

// 또는

SELECT ENAME, SAL, JOB
FROM emp
WHERE (job='SALESMAN'or job='ANALYST'or job='MANAGER');

```

배운 방법인 or은 촌스러운 방법이라면 다른 방법으로 검색해서 아래와 같이

▼ IN 기법을 사용.

1	SELECT	ENAME,SAL,JOB
2	FROM	emp
3	WHERE	job IN ('SALESMAN','ANALYST','MANAGER');

emp (9r × 3c)		
ENAME	SAL	JOB
BLAKE	2,850	MANAGER
CLARK	2,450	MANAGER
JONES	2,975	MANAGER
MARTIN	1,250	SALESMAN
ALLEN	1,600	SALESMAN
TURNER	1,500	SALESMAN
WARD	1,250	SALESMAN
FORD	3,000	ANALYST
SCOTT	3,000	ANALYST

```

SELECT ENAME, SAL, JOB
FROM emp
WHERE job IN ('SALESMAN','ANALYST','MANAGER');

```

+) 기타

- . 이중슬릿 실험 : 양자역학 관련.