2021-09-07

<계획>

책과 구글링, 강의를 통해 빠르게 익히고 적용시켜 9월 중으로 전체적인 구조파악 및 Springboot를 다루고 JPA까지도 이용해보는 과정을 가질 계획이다.

연휴가 있어서 충분히 할 수 있다고 생각한다.

Springboot JPA

<MySQL>

어제부터 MySQL 계속 공부하면서 개념을 정리하는 중.

SELECT \* FROM Customers;

여기서 \*는 Customers의 모든 컬럼을 말한다. 즉 Customers라는 테이블에서 모든 컬럼들을 가져오겠다. 라고 해석할 수 있다.

# SQL에서의 주석은 — 이다.

# 내가 원하는 테이블에서 원하는 컬럼만 가져오고 싶을 때는

SELECT 내가원하는컬럼 FROM 내가원하는테이블; -> 단 내가원하는컬럼은 내가원하는테이블 안에 있어야 한다.

내가원하는 컬럼이 여러가지 인 경우 ,찍고 원하는 컬럼을 쓰면 된다.

Q : 다른 테이블에 있는 각각 다른 컬럼들을 뽑아내고 싶을 때는 어떻게 해야할까?

# SELECT CustomerName, 1, 'Hello', NULL FROM Customers; 이렇게 입력하면

Customers 테이블에 있는 컬럼인 CustomerName을 읽어오고 그 옆에는 1로만 되어있는 컬럼, Hello로만 되어있는 컬럼, 비어있는 컬럼이 생긴다.

Hello에 ‘’를 붙인 이유는 문자열을 불러올 때 ‘’를 쓴다.

# MySQL에서 WHERE은 조건에 해당한다.

WHERE을 쓰게 되면 column(열)이 아니라 row(행)을 가져오게 된다.

# 원하는 순서로 데이터를 가져오고 싶을 때는 ORDER BY를 쓴다.

ASC는 오름차순(기본값이 오름차순이다) DESC는 내림차순.

Ex) SELECT \* FROM OrderDetails

ORDER BY ProductId ASC, Quantity DESC;

ORDER BY : 특정 기준 순서대로 값을 정렬하고 그 값을 기준으로 가져온다.

=> OrderDetaiils라는 테이블에서 모든 컬럼 중 ProductId는 오름차순으로,

같은 ProductId 끼리 중 Quantity는 내림차순으로 정렬 된 값을 기준으로 가져온다.

=> ProductId를 먼저 불러오고 그 다음 Quantity를 불러온다.

# 내가 원하는 만큼만 데이터를 가져오기 => LIMIT을 사용한다.

Ex) SELECT \* FROM Customers

LIMIT 10;

=> 10개만 가져온다. 0~10

LIMIT 0, 10; 이면 첫번째 나온 숫자는 건너 뛸 개수, 두번째 숫자는 가져올 개수를 말한다.

=> 즉 LIMIT 30, 10; 을 하면 31부터 10개 31~40까지 나온다.

=> 이런 LIMIT은 구글같은 페이지에서 페이지 단위 묶을 때 ( 페이징에 사용된다.)

# 원하는 별명으로 가져오기

AS를 사용해서 컬럼명을 변경할 수 있다.

SELECT

CustomerId AS ID,

CustomerName AS NAME,

Address AS ADDR

FROM Customers;

SELECT

CustomerId AS '아이디',

CustomerName AS '고객명',

Address AS '주소'

FROM Customers;

이런 식으로 변경할 수 있다.

SELECT 5 - 2.5 AS DIF;

=> 5 - 2.5의 결과값인 2.5를 DIF 라는 컬럼명으로 가져오겠다. 라는 뜻.

SELECT ‘ABC’ + 3;

=> 3이라는 결과값이 나온다 => MySQL에서는 문자열을 0으로 인식하여 0 + 3. 3이 나오게 된 것.

SELECT ‘1’ + ‘002’ \* 3;

=> 7이 나온다.

=> 숫자로 구성된 문자열은 숫자로 인식하여 계산한다는 뜻!

—————————

# 참, 거짓 관련 연산자 # ( 중요하다 )

SELECT TRUE, FALSE:

=> 1, 0 => TRUE는 1, FALSE는 0으로 나타내어진다.

WHERE을 쓰면 WHERE 뒤에 오는 조건이 맞는 경우에 그 값이 나타나게 된다.

따라서 WHERE 뒤에 참 거짓 연산자인 TRUE나 FALSE가 많이 사용하게 된다.

!TRUE = FALSE 이고 !FALSE = TRUE이다.

IS 연산자 : 양쪽이 모두 TRUE 또는 양쪽이 모두 FALSE

=> 즉 양쪽이 모두 같아야 TRUE 값을 반환한다.

AND && : 양쪽이 모두 TRUE일 때만 TRUE

OR || : 한쪽이 TRUE일 때만 TRUE

= : 양쪽값이 같다는 뜻.

다른 곳에서는 ==를 써야 하는데 MySQL에서는 = 한번만 써도 된다.

!=, <> : 양쪽 값이 다름.

>=, <= : (왼쪽, 오른쪽) 값이 같거나 더 큼.

문자열에서 부등호를 사용했을 때 더 나중에 오는가? 를 의미한다.

MySQL의 기본 사칙연산은 대소문자를 구분하지 않는다.

IN : 괄호 안의 값들 가운데에 있음.

NOT IN : 괄호 안의 값들 가운데에 없음.

LIKE연산자 : LIKE ‘ % ‘ : 0~N개 문자를 가진 패턴

LIKE ‘ ‘ : \_ 갯수만큼의 문자를 가진 패턴

——————————

# 숫자 관련 함수들

## 함수니까 ()가 붙는다!

ROUND : 반올림 CEIL : 올림 FLOOR : 내림

Ex) ROUND() CEIL() FLOOR()

ABS : 절댓값

GREATEST : 괄호 안에서 가장 큰 값

LEAST : 괄호 안에서 가장 작은 값

MAX, MIN은 괄호 안이 아닌 그것에 해당하는 컬럼을 기준으로 한다.

POW(A,B) : A를 B만큼 제곱

SQRT : 제곱근

TRUNCATE(N, n) : N을 소숫점 n자리까지 선택

UCASE, UPPER : 모두 대문자로

LCASE, LOWER : 모두 소문자로

CONCAT(…) : 괄호 안의 내용 이어붙임

CONCAT에서는 숫자도 문자로 변환되어서 그대로 들어가게 된다.

CONCAT\_WS() : 맨앞에 나오는 것을 그 다음에 나오는거 사이 사이에 넣는다.

LEFT() : 왼쪽에서 잘라낸다는 뜻

RIGHT() : 오른쪽에서 잘라낸다는 뜻

LENGTH : 문자열의 바이트 길이

CHAR\_LENGTH : 문자열의 문자 길이

TRIM : 양쪽 공백 제거

LTRIM : 왼쪽 공백 제거

RTRIM : 오른쪽 공백 제거

REPLACE(S,A,B) : S 중 A를 B로 바꾸겠다.라는 뜻

———————————————————————————————————————————————————————————————

# 시간 / 날짜 관련 기타 함수들

CURRENT\_DATE, CURDATE : 현재 날짜 반환

CURRENT\_DATE, CURTIME : 현재 시간 반환

CURRENT\_TIMESTAMP, NOW : 현재 시간과 날짜 반환

Ex) CURRENT\_DATE(), CURDATE(); 이런 식으로!

CURRENT\_DATE, CURDATE, CURTIME, NOW 다 따로 쓸 수 있다.

DATE : 문자열에 따라 날짜 생성

TIME : 문자열에 따라 시간 생성

Ex) DATE(‘2021-09-06’) : 2021 9월 6일로 읽는다.

YEAR() : 주어진 DATETIME 값의 년도 반환

LASTDAY();

|  |  |
| --- | --- |
| **%Y** | 년도 4자리 |
| **%y** | 년도 2자리 |
| **%M** | 월 영문 |
| **%m** | 월 숫자 |
| **%D** | 일 영문(1st, 2nd, 3rd...) |
| **%d**, **%e** | 일 숫자 (01 ~ 31) |
| **%T** | hh:mm:ss |
| **%r** | hh:mm:ss AM/PM |
| **%H**, **%k** | 시 (~23) |
| **%h**, **%l** | 시 (~12) |
| **%i** | 분 |
| **%S**, **%s** | 초 |
| **%p** | AM/PM |

DATE\_DIFF() : 두 시간/날짜 간 일수차

TIME\_DIFF() : 두 시간/날짜 간 시간차

LAST\_DAY() : 해당 달의 마지막 날짜

|  |  |
| --- | --- |
| **함수** | **설명** |
| **STR \_ TO \_ DATE**(S, F) | S를 F형식으로 해석하여 시간/날짜 생성 |

SELECT

OrderDate,

DATEDIFF(

STR\_TO\_DATE('1997-01-01 13:24:35', '%Y-%m-%d %T'),

OrderDate

),

TIMEDIFF(

STR\_TO\_DATE('1997-01-01 13:24:35', '%Y-%m-%d %T'),

STR\_TO\_DATE(CONCAT(OrderDate, ' ', '00:00:00'), '%Y-%m-%d %T')

)

FROM Orders;

|  |  |
| --- | --- |
| **형식** | **설명** |
| **IF**(조건, T, F) | 조건이 참이라면 T, 거짓이면 F 반환 |

SELECT IF (1 > 2, '1는 2보다 크다.', '1은 2보다 작다.');

복잡한 조건은 CASE를 사용한다.

SELECT

CASE

WHEN -1 > 0 THEN '-1은 양수다.'

WHEN -1 = 0 THEN '-1은 0이다.'

ELSE '-1은 음수다.'

END;

또는

SELECT

Price,

IF (Price > 30, 'Expensive', 'Cheap'),

CASE

WHEN Price < 20 THEN '저가'

WHEN Price BETWEEN 20 AND 30 THEN '일반'

ELSE '고가'

END

FROM Products;

이런 식으로.

|  |  |
| --- | --- |
| **형식** | **설명** |
| **IFNULL**(A, B) | A가 NULL일 시 B 출력 |

SELECT

IFNULL('A', 'B'),

IFNULL(NULL, 'B');

——————————————————————

[그룹 함수]

|  |  |
| --- | --- |
| **함수** | **설명** |
| **MAX** | 가장 큰 값 |
| **MIN** | 가장 작은 값 |
| **COUNT** | 갯수 (NULL값 제외) |
| **SUM** | 총합 |
| **AVG** | 평균 값 |

**GROUP BY - 조건에 따라 집계된 값을 가져옵니다.**

SELECT Country FROM Customers

GROUP BY Country;

**=> country끼리 같은 걸로 묶는다는 뜻**

<oding Test>

이것이 코딩테스트다 그리디 1 내가 짠 코드가 왜 오류인지 게속 고민하다 우연히 생각나서 다시 봤는데

n, k = map(int, input().split())

result = 0

while True:

if n % k == 0:

n = n // k

else:

n = n - 1

result += 1

if n == 1:

break

print(result)

else: 밑에

n = n – 1 을 그냥 n – 1로만 적어놨었다.

다시는 이런 실수 반복하면 안되겠다.

<OS>

운영체제의 인터럽트를 공부하던 도중 하드웨어 인터럽트는 정확하게 이해가 되는데 소프트웨어 인터럽트는 정확하게 이해가 되지 않는다.

가장 이해가 되지 않는 부분은

중간중간에 파면 팔수록 INT나 x86 z80과 같은 용어들과 마이크로프로세서의 개념까지 나오니 명확하게 한번에 정리되지 않는 느낌이다.

기본적인 개념 자체는 이해를 했으나 전체적으로 돌아가는 과정과 그 사이사이에 나오는 용어 및 개념들을 추가로 정리하면서 개념을 이해해야 할 것 같다.

<https://debonair-duck-f67.notion.site/8564b86e3c9b44a184b3aee67362def8>

스스로 내용을 찾아보면서 notion에 정리를 하는 중.