06

다양한 SQL 함수 사용하기

06-1 문자열 함수

- 문자열 함수는 문자열 연결, 형 변환, 공백 제거, 치환 등에서 다양하게 사용함
- 문자열 함수는 문자열뿐만 아니라 열 이름을 인자로 활용할 수 있음

▪ 문자열과 문자열을 연결하는 함수 — CONCAT

- 연결할 문자열을 괄호 안에 쉼표로 구분하여 함수의 인자로 나열하면 됨
- 단어가 아닌 열 이름을 나열할 때도 마찬가지임

CONCAT 함수 예

```
SELECT CONCAT('I ', 'Love ', 'MySQL') AS col_1;
```

06-1 문자열 함수

▪ 문자열과 문자열을 연결하는 함수 — CONCAT

1. 다음은 열 이름과 문자열을 인자로 전달하는 쿼리임.

Do it! CONCAT 함수로 열 이름과 문자열 연결

```
SELECT CONCAT(first_name, ', ', last_name) AS customer_name FROM customer;
```

first_name과 last_name 열을 연결할 때
이를 구분하기 위해 중간에 문자열인 ', '를 사용

실행 결과

	customer_name
▶	MARY, SMITH
	PATRICIA, JOHNSON
	LINDA, WILLIAMS
	BARBARA, JONES
	ELIZABETH, BROWN
	JENNIFER, DAVIS
	MARIA, MILLER
	SUSAN, WILSON
	MARGARET, MOORE
	DOROTHY, TAYLOR
	LISA, ANDERSON
	NANCY, THOMAS
	KAREN, JACKSON
	BETTY, WHITE
	HELEN, HARRIS

06-1 문자열 함수

▪ 문자열과 문자열을 연결하는 함수 — CONCAT

2. 여러 열을 합칠 때 모든 열 이름 사이에 직접 일일이 쉼표를 입력하기는 번거로움. 이때 CONCAT_WS를 사용하면 구분자를 미리 정의해서 자동으로 적용할 수 있음.

Do it! CONCAT_WS 함수로 구분자 지정

```
SELECT CONCAT_WS(', ', first_name, last_name, email) AS customer_name FROM customer;
```

실행 결과

customer_name
MARY, SMITH, MARY.SMITH@sakilacustomer.org
PATRICIA, JOHNSON, PATRICIA.JOHNSON@sakilacustomer.org
LINDA, WILLIAMS, LINDA.WILLIAMS@sakilacustomer.org
BARBARA, JONES, BARBARA.JONES@sakilacustomer.org
ELIZABETH, BROWN, ELIZABETH.BROWN@sakilacustomer.org
JENNIFER, DAVIS, JENNIFER.DAVIS@sakilacustomer.org
MARIA, MILLER, MARIA.MILLER@sakilacustomer.org
SUSAN, WILSON, SUSAN.WILSON@sakilacustomer.org
MARGARET, MOORE, MARGARET.MOORE@sakilacustomer.org
DOROTHY, TAYLOR, DOROTHY.TAYLOR@sakilacustomer.org
LISA, ANDERSON, LISA.ANDERSON@sakilacustomer.org
NANCY, THOMAS, NANCY.THOMAS@sakilacustomer.org
KAREN, JACKSON, KAREN.JACKSON@sakilacustomer.org
BETTY, WHITE, BETTY.WHITE@sakilacustomer.org
HELEN, HARRIS, HELEN.HARRIS@sakilacustomer.org

▪ 문자열과 문자열을 연결하는 함수 — CONCAT

3. 만약 인수로 NULL을 입력할 경우 결과는 모두 NULL이 반환됨.
이는 NULL이 어떠한 문자열과 결합하거나 계산되더라도 결과가 NULL이기 때문임

Do it! CONCAT 함수로 NULL과 열 이름 연결

```
SELECT CONCAT(NULL, first_name, last_name) AS customer_name
FROM customer;
```

실행 결과

customer_name
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL

▪ 문자열과 문자열을 연결하는 함수 — CONCAT

4. CONCAT_WS는 비어 있는 열을 그대로 적용함.
그러나 구분 인수 뒤에 오는 NULL은 무시하여 건너뜀.

Do it! CONCAT_WS 인자로 NULL이 있는 경우

```
SELECT CONCAT_WS(', ', first_name, NULL, last_name) as customer_name
FROM customer;
```

실행 결과

customer_name
MARY, SMITH
PATRICIA, JOHNSON
LINDA, WILLIAMS
BARBARA, JONES
ELIZABETH, BROWN
JENNIFER, DAVIS
MARIA, MILLER
SUSAN, WILSON
MARGARET, MOORE
DOROTHY, TAYLOR
LISA, ANDERSON
NANCY, THOMAS
KAREN, JACKSON
BETTY, WHITE
HELEN, HARRIS

06-1 문자열 함수

▪ 데이터 형 변환 함수 — CAST, CONVERT

- 명시적 형 변환

CAST 함수의 기본 형식

```
SELECT CAST(열 AS 데이터 유형) FROM 테이블;
```

CONVERT 함수의 기본 형식

```
SELECT CONVERT(열, 데이터 유형) FROM 테이블;
```

- CONVERT 함수는 CAST 함수와 달리
  문자열 집합을 다른 문자열 집합으로 변환할 수 있음

06-1 문자열 함수

▪ 데이터 형 변환 함수 — CAST, CONVERT

- CAST와 CONVERT를 이용해 형 변환을 할 수 있는 데이터 유형은 한정적임

CAST 및 CONVERT에 사용 가능한 데이터 유형

BINARY	CHAR
DATE	DATETIME
TIME	DECIMAL
JSON	NCHAR
SIGNED[INTEGER]	UNSIGNED[INTEGER]

06-1 문자열 함수

▪ 데이터 형 변환 함수 — CAST, CONVERT

1.

Do it! 문자열을 부호 없는 정수형으로 변경

```
SELECT
        4 / '2',
        4 / 2,
        4 / CAST('2' AS UNSIGNED);
```

실행 결과

	4 / '2'	4 / 2	4 / CAST('2' AS UNSIGNED)
▶	2	2.0000	2.0000

▪ 데이터 형 변환 함수 — CAST, CONVERT

2. 날짜형 데이터를 숫자형으로 변환

Do it! NOW 함수로 현재 날짜와 시간 출력

```
SELECT NOW();
```

실행 결과

	NOW()
▶	2023-08-20 13:29:46

Do it! CAST 함수로 날짜형을 숫자형으로 변환

```
SELECT CAST(NOW() AS SIGNED);
```

실행 결과

	CAST(NOW() AS SIGNED)
▶	20230820133013

▪ 데이터 형 변환 함수 — CAST, CONVERT

3. 반대로 숫자형을 날짜형 또는 문자열로 변환해 보자.
마찬가지로 CAST 함수를 활용함

Do it! CAST 함수로 숫자형을 날짜형으로 변환

```
SELECT CAST(20230819 AS DATE);
```

실행 결과

	CAST(20230819 AS DATE)
▶	2023-08-19

Do it! CAST 함수로 숫자형을 문자열로 변환

```
SELECT CAST(20230819 AS CHAR);
```

실행 결과

	CAST(20230819 AS CHAR)
▶	20230819

06-1 문자열 함수

▪ 데이터 형 변환 함수 — CAST, CONVERT

4. CONVERT 함수는 다음과 같은 형태로 인자 2개를 넘겨 사용함.

Do it! CONVERT 함수로 날짜형을 정수형으로 변환

```
SELECT CONVERT(NOW(), SIGNED);
```

실행 결과

	CONVERT(now(), SIGNED)
▶	20230820133950

▪ 데이터 형 변환 함수 — CAST, CONVERT

5. CONVERT 함수를 사용해
숫자형으로 된 데이터를 날짜형 데이터로 변환

Do it! CONVERT 함수로 숫자형을 날짜형으로 변환

```
SELECT CONVERT(20230819, DATE);
```

실행 결과

	CONVERT(20230819, DATE)
▶	2023-08-19

▪ 데이터 형 변환 함수 — CAST, CONVERT

6. CAST, CONVERT 함수를 사용할 때
AS CHAR(5) 또는 CHAR(5)와 같이 문자열의 길이를 지정할 수도 있음.

Do it! CHAR로 데이터 길이 지정

```
SELECT CONVERT(20230819, CHAR(5));
```

실행 결과

	CONVERT(20230819, CHAR(5))
▶	20230

▪ 데이터 형 변환 함수 — CAST, CONVERT

7. 엄청난 큰 수에 1을 더하는 쿼리를 작성해 보자.
이를 실행하면 가장 큰 숫자형(BIGINT)의 범위를 넘어 오버플로가 발생함.

Do it! 오버플로 발생 예

```
SELECT 92233720368547758O7 + 1;
```

```
Error Code: 1690. BIGINT value is out of range in '(92233720368547758O7 + 1)'
```

▪ 데이터 형 변환 함수 — CAST, CONVERT

7. 오버플로를 예방하고 싶다면
CAST, CONVERT 함수를 활용해 입력값을 UNSIGNED로 변경하여 연산하면 됨.

Do it! CAST 함수로 오버플로 방지

```
SELECT CAST(92233720368547758O7 AS UNSIGNED) + 1;
```

실행 결과

	CAST(9223372036854775807 AS UNSIGNED) + 1
▶	9223372036854775808

Do it! CONVERT함수로 오버플로 방지

```
SELECT CONVERT(9223372036854775807, UNSIGNED) + 1;
```

실행 결과

	CONVERT(9223372036854775807,UNSIGNED) + 1
▶	9223372036854775808

06-1 문자열 함수

▪ NULL을 대체하는 함수 — IFNULL, COALESCE

- NULL은 어떠한 연산 작업을 진행해도 NULL이 반환되므로
  NULL 때문에 예기치 못한 결과를 얻게 되는 경우가 종종 있음
- 테이블에 NULL이 있는 경우 문자열 또는 숫자로 데이터를 바꾸는 것이 좋음

- IFNULL 함수 : 인자로 열 이름과 대체할 값을 전달하면 해당 열에 있는 NULL을 대체할 값으로 변환함

IFNULL 함수의 기본 형식

IFNULL(열, 대체할 값)

- COALESCE는 NULL이 아닌 값이 나올 때까지 후보군의 여러 열을 입력할 수 있음

COALESCE 함수의 기본 형식

COALESCE(열 1, 열 2, ...)

06-1 문자열 함수

▪ NULL을 대체하는 함수 — IFNULL, COALESCE

1. doit_null이라 테이블을 생성하는 쿼리를 작성해 보자. 이 테이블에는 NULL이 존재함

Do it! 테이블 생성

```
CREATE TABLE doit_null (
col_1 INT,
col_2 VARCHAR(10),
col_3 VARCHAR(10),
col_4 VARCHAR(10),
col_5 VARCHAR(10)
);

INSERT INTO doit_null VALUES (1, NULL, 'col_3', 'col_4', 'col_5');
INSERT INTO doit_null VALUES (2, NULL, NULL, NULL, 'col_5');
INSERT INTO doit_null VALUES (3, NULL, NULL, NULL, NULL);

SELECT * FROM doit_null;
```

실행 결과

	col_1	col_2	col_3	col_4	col_5
▶	1	NULL	col_3	col_4	col_5
	2	NULL	NULL	NULL	col_5
	3	NULL	NULL	NULL	NULL

06-1 문자열 함수

▪ NULL을 대체하는 함수 — IFNULL, COALESCE

1. 먼저 col_2 열의 값이 NULL이면 IFNULL 함수를 사용해 공백('')으로 대체

Do it! IFNULL 함수로 col_2열의 NULL 대체

```
SELECT col_1, IFNULL(col_2, '') AS col_2, col_3, col_4, col_5
FROM doit_null WHERE col_1 = 1;
```

실행 결과

	col_1	col_2	col_3	col_4	col_5
▶	1		col_3	col_4	col_5

▪ NULL을 대체하는 함수 — IFNULL, COALESCE

3. col_2열의 값이 NULL이면 col_3 열의 값으로 대체해 보자.

Do it! IFNULL 함수로 col_3열의 NULL 대체

```
SELECT col_1, IFNULL(col_2, col_3) AS col_2, col_3, col_4, col_5
FROM doit_null WHERE col_1 = 1;
```

실행 결과

	col_1	col_2	col_3	col_4	col_5
▶	1	col_3	col_3	col_4	col_5

▪ NULL을 대체하는 함수 — IFNULL, COALESCE

4. COALESCE 함수는 첫 번째 인자로 전달한 열에 NULL이 있을 때
그 다음 인자로 작성한 열의 데이터로 대체함.
만약 이 함수에 N개의 인자를 작성했다면 순차로 대입함.

Do it! COALESCE 함수로 NULL을 다른 데이터로 대체: 마지막 인자에 데이터가 있는 경우

```
SELECT col_1, COALESCE(col_2, col_3, col_4, col_5) FROM doit_null WHERE col_1 = 2;
```

col_2의 값이 NULL일 때
다음 인자인 col_3의 데이터도 NULL이면
그다음 인자인 col_4의 데이터를 확인하여 대입함

실행 결과

	col_1	COALESCE(col_2, col_3, col_4, col_5)
▶	3	col_5

▪ NULL을 대체하는 함수 — IFNULL, COALESCE

5. 마지막 인자까지도 NULL이 저장되어 있다면 결국 NULL을 반환함

Do it! COALESCE 함수로 NULL을 다른 데이터로 대체: 마지막 인자에도 NULL이 있는 경우

```
SELECT col_1, COALESCE(col_2, col_3, col_4, col_5)  FROM doit_null WHERE col_1 = 3;
```

실행 결과

	col_1	COALESCE(col_2, col_3, col_4, col_5)
▶	3	NULL

06-1 문자열 함수

▪ 소문자 혹은 대문자로 변경하는 함수 — LOWER, UPPER

1. LOWER 함수는 대문자를 소문자로, UPPER 함수는 소문자를 대문자로 변경함.

Do it! LOWER 함수로 소문자를, UPPER 함수로 대문자로 변경

```
SELECT 'Do it! SQL', LOWER('Do it! SQL'), UPPER('Do it! SQL');
```

실행 결과

	Do it! SQL	LOWER('Do it! SQL')	UPPER('Do it! SQL')
▶	Do it! SQL	do it! sql	DO IT! SQL

06-1 문자열 함수

▪ 소문자 혹은 대문자로 변경하는 함수 — LOWER, UPPER

2. 문자열 대신 열 이름을 입력해 보자.
해당 열의 데이터들이 소문자 또는 대문자로 변경되어 출력된 것을 확인할 수 있음

Do it! LOWER 함수로 소문자를, UPPER 함수로 대문자로 변경

```
SELECT email, LOWER(email), UPPER(email) FROM customer;
```

실행 결과

email	LOWER(email)	UPPER(email)
MARY.SMITH@sakilacustomer.org	mary.smith@sakilacustomer.org	MARY.SMITH@SAKILACUSTOMER.ORG
PATRICIA.JOHNSON@sakilacustomer.org	patricia.johnson@sakilacustomer.org	PATRICIA.JOHNSON@SAKILACUSTOMER.ORG
LINDA.WILLIAMS@sakilacustomer.org	linda.williams@sakilacustomer.org	LINDA.WILLIAMS@SAKILACUSTOMER.ORG

06-1 문자열 함수

▪ 공백을 제거하는 함수 — LTRIM, RTRIM, TRIM

- 사용자가 어떤 데이터를 입력할 때 공백을 입력하는 경우가 있는데,
  데이터 관리자에게는 공백이 문제를 일으킬 수 있으므로 관리 대상임

- 예를 들어 사용자가 회원 가입을 할 때 실수로 아이디나 비밀번호 뒤에
  의도하지 않은 공백을 입력했고 데이터베이스에서 공백을 허용했다면
  이후 사용자가 로그인할 때 아이디, 비밀번호를 제대로 입력해도
  공백이 입력이 되지 않아 인증 처리가 제대로 되지 않을 것임

06-1 문자열 함수

▪ 공백을 제거하는 함수 — LTRIM, RTRIM, TRIM

- LTRIM 함수의 L은 왼쪽을, RTRIM 함수의 R은 오른쪽을 의미하며 각각 문자열의 왼쪽(앞) 또는 오른쪽(뒤)의 공백을 제거함
- 양쪽 공백을 모두 제거하려면 TRIM 함수를 사용하면 됨

1. LTRIM을 사용해 'Do it!' 문자열 앞 공백을 삭제

Do it! LTRIM 함수로 왼쪽 공백 제거

```
SELECT ' Do it! MySQL', LTRIM(' Do it! MySQL');
```

실행 결과

	Do it! MySQL	LTRIM(' Do it! MySQL')
▶	Do it! MySQL	Do it! MySQL

▪ 공백을 제거하는 함수 — LTRIM, RTRIM, TRIM

2. RTRIM을 사용해 'Do it!' 문자열 뒤 공백을 삭제

Do it! RTRIM 함수로 오른쪽 공백 제거

```
SELECT 'Do it! MySQL ', RTRIM('Do it! MySQL ');
```

실행 결과

	Do it! MySQL	RTRIM('Do it! MySQL ')
▶	Do it! MySQL	Do it! MySQL

3. TRIM 함수를 사용해 문자열의 양쪽 공백을 모두 제거

Do it! TRIM 함수로 양쪽 공백 제거

```
SELECT ' Do it! MySQL ', TRIM(' Do it! MySQL ');
```

실행 결과

	Do it! MySQL	TRIM(' Do it! MySQL ')
▶	Do it! MySQL	Do it! MySQL

▪ 공백을 제거하는 함수 — LTRIM, RTRIM, TRIM

4. TRIM 함수는 공백이 아닌 단어 앞뒤에 있는 특정 문자를 제거하는 기능도 있음.

Do it! TRIM 함수로 양쪽 문자 제거

```
SELECT TRIM(BOTH '#' FROM '# Do it! MySQL #');
```

실행 결과

	TRIM(BOTH '#' FROM '# Do it! SQL #')
▶	Do it! SQL

LEADING 을 입력하면 왼쪽문자가, TRAILING을 입력하면 오른쪽 문자가 제거됨

06-1 문자열 함수

▪ 문자열 크기 또는 개수를 반환하는 함수 — LENGTH, CHAR_LENGTH

1. 영어 문자열과 한글 문자열의 바이트 크기를 확인할 수 있음.

Do it! LENGTH 함수로 문자열의 크기 반환

```
SELECT LENGTH('Do it! MySQL'), LENGTH('두잇 마이에스큐엘');
```

실행 결과

	LENGTH('Do it! MySQL')	LENGTH('두잇 마이에스큐엘')
▶	12	25

LENGTH 함수를 사용해 'Do it! MySQL'의 문자열 크기로는 12바이트를,
'두잇 마이에스큐엘'의 문자열 크기로는 25바이트를 반환한 것을 확인할 수 있음

06-1 문자열 함수

▪ 문자열 크기 또는 개수를 반환하는 함수 — LENGTH, CHAR_LENGTH

2. 영어 및 공백은 1바이트, 한글과 한자는 3바이트, 특수문자는 3바이트를 사용함

Do it! LENGTH 함수로 다양한 문자의 크기 반환

```
SELECT LENGTH('A'), LENGTH('강'), LENGTH('漢'), LENGTH('◁'), LENGTH(' ');
```

실행 결과

	LENGTH('A')	LENGTH('강')	LENGTH('漢')	LENGTH('◁')	LENGTH(' ')
▶	1	3	3	3	1

06-1 문자열 함수

▪ 문자열 크기 또는 개수를 반환하는 함수 — LENGTH, CHAR_LENGTH

3. 바이트 크기로는 문자열의 개수를 정확히 알기 어려움.
문자열 개수를 확인하고 싶다면 CHAR_LENGTH 함수를 사용하면 됨.

Do it! CHAR_LENGTH 함수로 문자열의 개수 반환

```
SELECT CHAR_LENGTH('Do it! MySQL'), CHAR_LENGTH('두잇 마이에스큐엘');
```

실행 결과

	CHAR_LENGTH('Do it! MySQL')	CHAR_LENGTH('두잇 마이에스큐엘')
▶	12	9

CHAR_LENGTH 함수를 사용해 'Do it! MySQL'의 문자열 개수로는
띄어쓰기를 포함해 12, '두잇 마이에스큐엘'은 9를 반환한 것을 확인할 수 있음

06-1 문자열 함수

▪ 문자열 크기 또는 개수를 반환하는 함수 — LENGTH, CHAR_LENGTH

4. LENGTH와 CHAR_LENGTH 함수에 문자열 대신 열 이름을 인수로 전달할 수도 있음.

Do it! LENGTH와 CHAR_LENGTH 함수에 열 이름 전달

```
SELECT first_name, LENGTH(first_name), CHAR_LENGTH(first_name) FROM customer;
```

실행 결과

	first_name	LENGTH(first_name)	CHAR_LENGTH(first_name)
	CHRISTINE	9	9
	MARIE	5	5
	JANET	5	5
	CATHERINE	9	9
	FRANCES	7	7
	ANN	3	3
	JOYCE	5	5
	DIANE	5	5
	ALICE	5	5
	JULIE	5	5

06-1 문자열 함수

▪ 특정 문자까지의 문자열 길이를 반환하는 함수 — POSITION

1. 느낌표가 있는 문자까지의 문자열 **길이**를 반환 -> **위치**라는 표현이 더 적절

Do it! POSITION 함수로 특정 문자까지의 크기 반환

```
SELECT 'Do it!! SQL', POSITION('!' IN 'Do it!! MySQL');
```

실행 결과

	Do it!! SQL	POSITION('!' IN 'Do it!! SQL')
▶	Do it!! SQL	6

2. 지정한 문자가 탐색 대상이 되는 문자열에 존재하지 않으면 0을 반환

Do it! 탐색 문자가 없는 경우

```
SELECT 'Do it!! SQL', POSITION('#' IN 'Do it!! MySQL');
```

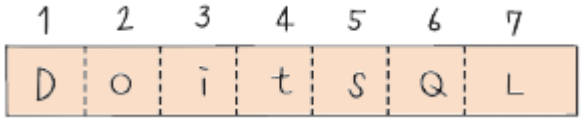
실행 결과

	Do it!! SQL	POSITION('#' IN 'Do it!! SQL')
▶	Do it!! SQL	0

06-1 문자열 함수

▪ 지정한 길이만큼 문자열을 반환하는 함수 — LEFT, RIGHT

- LEFT 함수는 문자열의 왼쪽부터,
  RIGHT 함수는 오른쪽부터 정의한 위치만큼의 문자열을 반환함



Do it! LEFT와 RIGHT 함수로 왼쪽과 오른쪽 2개의 문자열 반환

```
SELECT 'Do it! MySQL', LEFT('Do it! MySQL', 2), RIGHT('Do it! MySQL', 2);
```

실행 결과

	Do it! SQL	LEFT('Do it! SQL', 2)	RIGHT('Do it! SQL', 2)
▶	Do it! SQL	Do	QL

06-1 문자열 함수

▪ 지정한 범위의 문자열을 반환하는 함수 — SUBSTRING

- SUBSTRING 함수는 지정한 범위의 문자열을 반환함
- 함수의 2번째 인자에는 시작 위치를, 3번째 인자에는 시작 위치로부터 반환할 문자열 개수를 입력함

1. 다음은 4번째 문자부터 문자 2개를 반환하는 쿼리

Do it! SUBSTRING 함수로 지정한 범위의 문자열 반환

```
SELECT 'Do it! MySQL', SUBSTRING('Do it! MySQL', 4, 2);
```

실행 결과

	Do it! SQL	SUBSTRING('Do it! SQL', 4, 2)
▶	Do it! SQL	it

06-1 문자열 함수

▪ 지정한 범위의 문자열을 반환하는 함수 — SUBSTRING

2. SUBSTRING 함수도 다른 함수처럼 열 이름을 인수로 전달하여 사용할 수도 있음.

Do it! SUBSTRING 함수에 열 이름 전달

```
SELECT first_name, SUBSTRING(first_name, 2, 3) FROM customer;
```

실행 결과

	first_name	SUBSTRING(first_name, 2,3)
▶	MARY	ARY
	PATRICIA	ATR
	LINDA	IND
	BARBARA	ARB
	ELIZABETH	LIZ
	JENNIFER	ENN
	MARIA	ARI
	SUSAN	USA
	MARGARET	ARG

06-1 문자열 함수

▪ 지정한 범위의 문자열을 반환하는 함수 — SUBSTRING

3. SUBSTRING 함수는 POSITION 함수와 함께 사용할 수도 있음.
다음은 POSITION 함수로 @ 문자 위치를 계산한 다음,
SUBSTRING 함수에 사용하여 @ 문자 바로 앞까지의 문자열을 조회하는 쿼리임.

Do it! SUBSTRING과 POSITOIN 함수 조합

```
SELECT SUBSTRING('abc@email.com', 1, POSITION('@' IN 'abc@email.com') -1);
```

실행 결과

	SUBSTRING('abc@email.com', 1, POSITION('@' IN 'abc@email.com') -1)
▶	abc

06-1 문자열 함수

▪ 특정 문자를 다른 문자로 대체하는 함수 — REPLACE

- 지정한 문자를 다른 문자로 대체함
- customer 테이블의 first_name 열에서 A로 시작하는 데이터를 조회해 이 데이터에 있는 문자열 A를 C로 대체

Do it! REPLACE 함수로 문자 변경

```
SELECT first_name, REPLACE(first_name, 'A', 'C') FROM customer WHERE first_name LIKE 'A%';
```

실행 결과

	first_name	REPLACE(first_name, 'A', 'C')
▶	ANGELA	CNGELC
	AMY	CMY
	ANNA	CNNC
	AMANDA	CMCNDC
	ANN	CNN
	ALICE	CLICE
	ASHLEY	CSHLEY
	ANDREA	CNDREC
	ANNE	CNNE

06-1 문자열 함수

▪ 문자를 반복하는 함수 — REPEAT

- REPEAT 함수는 지정한 문자를 반복할 때 사용

1. 문자열 0을 10번 반복해 출력

Do it! REPEAT 함수로 문자 반복

```
SELECT REPEAT('0', 10);
```

실행 결과

	REPEAT('0', 10)
▶	0000000000

2. REPEAT 함수도 다른 함수와 조합하여 사용할 수 있음. %A%를 사용해 A가 포함된 모든 데이터를 조회한 뒤, 문자 A를 10개의 C로 대체한 것을 확인할 수 있음

Do it! REPEAT과 REPLACE 함수 조합

```
SELECT first_name, REPLACE(first_name, 'A', REPEAT('C', 10))
FROM customer WHERE first_name LIKE '%A%';
```

실행 결과

	first_name	REPLACE(first_name, 'A', REPEAT('C', 10))
▶	MARY	MCCCCCCCCCCRY
	PATRICIA	PCCCCCCCCCCTRICICCCCCCCCCC
	LINDA	LINDCCCCCCCCCC
	BARBARA	BCCCCCCCCCCRBCCCCCCCCCCRCCCCCCCCCC
	ELIZABETH	ELIZCCCCCCCCCCBETH
	MARIA	MCCCCCCCCCCRICCCCCCCCCC
	SUSAN	SUSCCCCCCCCCCN
	MARGARET	MCCCCCCCCCCRGCCCCCCCCCCRET
	LISA	LISCCCCCCCCCC
	NANCY	NCCCCCCCCCCNCY

06-1 문자열 함수

▪ 공백 문자를 생성하는 함수 — SPACE

- 지정한 인자만큼 공백을 생성

Do it! SPACE 함수로 공백 문자 반복

```
SELECT CONCAT(first_name, SPACE(10), last_name) FROM customer;
```

실행 결과

	CONCAT(first_name, SPACE(10), last_name)
▶	MARY SMITH
	PATRICIA JOHNSON
	LINDA WILLIAMS
	BARBARA JONES
	ELIZABETH BROWN
	JENNIFER DAVIS
	MARIA MILLER
	SUSAN WILSON
	MARGARET MOORE

06-1 문자열 함수

▪ 문자열을 역순으로 출력하는 함수 — REVERSE

- REVERSE 함수는 문자열을 거꾸로 정렬하는 함수임
- 사용할 일이 없을 것 같지만 다양한 문자열 함수와 혼합하면
  이메일에서 도메인의 자릿수나 IP 대역을 구할 때 등 다양하게 활용 할 수 있음

1. REVERSE 함수를 사용해 문자열을 역순으로 반환해 보자.

Do it! REVERSE 함수로 문자열을 역순으로 반환

```
SELECT 'Do it! MySQL', REVERSE('Do it! MySQL');
```

실행 결과

	Do it! SQL	reverse('Do it! SQL')
▶	Do it! SQL	LQS !ti oD

▪ 문자열을 역순으로 출력하는 함수 — REVERSE

2. REVERSE 함수도 여러 함수와 조합하여 사용하면 매우 유용함.
다음 쿼리는 POSITION 함수, CHAR_LENGTH 함수, SUBSTRING 함수를 조합하여 IP 주소의 3번째 부분까지 정보만 조회함

Do it! REVERSE 함수와 다른 여러 함수 조합

```
WITH ip_list (ip)
AS (
        SELECT '192.168.0.1' UNION ALL
        SELECT '10.6.100.99' UNION ALL
        SELECT '8.8.8.8' UNION ALL
        SELECT '192.200.212.113'
)
SELECT ip, SUBSTRING(ip, 1, CHAR_LENGTH(ip) - POSITION('.' IN REVERSE(ip)))
FROM ip_list;
```

실행 결과

ip	SUBSTRING(ip, 1, CHAR_LENGTH(ip) - POSITION('.' IN REVERSE(ip)))
192.168.0.1	192.168.0
10.6.100.99	10.6.100
8.8.8.8	8.8.8
192.200.212.113	192.200.212

06-1 문자열 함수

▪ 문자열을 비교하는 함수 — STRCMP

- STRCMP 함수는 두 문자열을 비교하여 동일한지를 알려 줌
- 비교하는 두 문자열이 동일할 경우 0을, 다를 경우 -1을 반환함

Do it! STRCMP 함수로 두 문자열을 비교: 동일한 경우

```
SELECT STRCMP('Do it! MySQL', 'Do it! MySQL');
```

실행 결과

	STRCMP('Do it! MySQL', 'Do it! MySQL')
▶	0

Do it! STRCMP 함수로 두 문자열을 비교: 동일하지 않은 경우

```
SELECT STRCMP('Do it! MySQL', 'Do it! MySQL!');
```

실행 결과

	STRCMP('Do it! MySQL', 'Do it! MySQL!')
▶	-1

06-2 날짜 함수

- 날짜 함수로는 날짜나 시간 데이터를 활용한 작업을 할 수 있음
- 날짜 함수는 일정 기간의 데이터를 검색할 때 빈번히 사용함
- 특히 같은 연, 월, 일, 요일 등 조건에 따라 데이터를 검색할 때 날짜 함수를 사용하면 매우 편리함

06-2 날짜 함수

▪ 서버의 현재 날짜나 시간을 반환하는 다양한 함수

1. 접속 중인 데이터베이스 서버의
현재 날짜를 확인하려면 CURRENT_DATE 함수를,
시간을 알고 싶으면 CURRENT_TIME 함수를
날짜와 시간을 합쳐 확인하고 싶다면 CURRENT_TIMESTAMP 또는 NOW 함수를 사용

Do it! 날짜 함수로 현재 날짜나 시간 반환

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

실행 결과

	CURRENT_DATE()	CURRENT_TIME()	CURRENT_TIMESTAMP()	NOW()
▶	2023-08-20	16:15:42	2023-08-20 16:15:42	2023-08-20 16:15:42

06-2 날짜 함수

▪ 서버의 현재 날짜나 시간을 반환하는 다양한 함수

2. 정밀한 시간을 확인하려면 밀리초나 마이크로초까지 필요할 수도 있음.
시간을 반환하는 함수에 인수로 3을 입력했는데, 이는 밀리초 단위까지 출력하도록 요청함

Do it! 정밀한 시각을 반환

```
SELECT CURRENT_DATE(), CURRENT_TIME(3), CURRENT_TIMESTAMP(3), NOW(3);
```

실행 결과

	CURRENT_DATE()	CURRENT_TIME(3)	CURRENT_TIMESTAMP(3)	NOW(3)
▶	2023-08-20	16:18:10.934	2023-08-20 16:18:10.934	2023-08-20 16:18:10.934

06-2 날짜 함수

▪ 서버의 현재 날짜나 시간을 반환하는 다양한 함수

3. UTC(Coordinated Universal Time)는 세계 표준 시간으로, 즉 국제 표준 시간의 기준으로 쓰이는 시각을 의미함.

Do it! UTC_DATE, UTC_TIME, UTC_TIMESTAMP 함수로 세계 표준 날짜나 시간 반환

```
SELECT CURRENT_TIMESTAMP(3), UTC_DATE(), UTC_TIME(3), UTC_TIMESTAMP(3);
```

실행 결과

	CURRENT_TIMESTAMP(3)	UTC_DATE()	UTC_TIME(3)	UTC_TIMESTAMP(3)
▶	2023-08-20 16:22:15.737	2023-08-20	07:22:15.737	2023-08-20 07:22:15.737

06-2 날짜 함수

▪ 날짜를 더하거나 빼는 함수 — DATE_ADD, DATE_SUB

1. 날짜를 더하거나 빼려면 DATE_ADD 함수를 사용함.

   DATE_ADD 함수는 첫 번째 인수로 날짜 데이터를 입력하고,
   두 번째 인자로 INTERVAL과 함께 더하거나 빼고자 하는 숫자
   그리고 연, 월, 일 등의 단위를 넣음.

Do it! DATE_ADD 함수로 1년 증가한 날짜 반환

```
SELECT NOW(), DATE_ADD(NOW(), INTERVAL 1 YEAR);
```

실행 결과

	NOW()	DATE_ADD(NOW(), INTERVAL 1 YEAR)
▶	2023-08-20 16:30:21	2024-08-20 16:30:21

▪ 날짜를 더하거나 빼는 함수 — DATE_ADD, DATE_SUB

2. 만약 원하는 만큼의 날짜를 뺄 때에는 음수를 입력함.

Do it! DATE_ADD 함수로 1년 감소한 날짜 반환

```
SELECT NOW(), DATE_ADD(NOW(), INTERVAL -1 YEAR);
```

실행 결과

	NOW()	DATE_ADD(NOW(), INTERVAL -1 YEAR)
▶	2023-08-20 16:43:52	2022-08-20 16:43:52

06-2 날짜 함수

▪ 날짜를 더하거나 빼는 함수 — DATE_ADD, DATE_SUB

2. DATE_ADD 함수에서는 더하거나 빼는 숫자와 함께 사용하는 단위로 YEAR뿐만 아니라 다른 시간 단위도 사용할 수 있음

단위	의미
YEAR	년
QUARTER	분기
MONTH	월
DAY	일
WEEK	주
HOUR	시간
MINUTE	분
SECOND	초
MICROSECOND	마이크로초

▪ 날짜를 더하거나 빼는 함수 — DATE_ADD, DATE_SUB

3. 날짜를 뺄 때에는 DATE_ADD 함수에 -1과 같은 음수를 입력했지만 DATE_SUB 함수를 사용해도 원하는 만큼 날짜를 뺄 수 있음.

   단, 이때는 숫자를 양수로 입력해야 함. 음수로 입력하면 오히려 날짜를 더하게 됨.

Do it! DATE_SUB 함수로 1년 감소한 날짜 반환

```
SELECT NOW(), DATE_SUB(NOW(), INTERVAL 1 YEAR), DATE_SUB(NOW(), INTERVAL -1 YEAR);
```

실행 결과

	NOW()	DATE_SUB(NOW(), INTERVAL 1 YEAR)	DATE_SUB(NOW(), INTERVAL -1 YEAR)
▶	2023-08-20 16:49:14	2022-08-20 16:49:14	2024-08-20 16:49:14

06-2 날짜 함수

▪ 날짜 간 차이를 구하는 함수 — DATEDIFF, TIMESTAMPDIFF

- DATEDIFF 함수는 날짜 간의 시간 차이를 구할 수 있음

1. 2023년 12월 31일에서 2023년 01월 01일까지의 일수 차이는 365일로 출력

Do it! DATEDIFF 함수로 날짜 간의 일수 차 반환

```
SELECT DATEDIFF('2023-12-31 23:59:59.9999999', '2023-01-01 00:00:00.0000000');
```

실행 결과

	DATEDIFF('2023-12-31 23:59:59.9999999', '2023-01-01 00:00:00.0000000')
▶	365

▪ 날짜 간 차이를 구하는 함수 — DATEDIFF, TIMESTAMPDIFF

2. 일수가 아닌 년 또는 시간 등 다양한 단위로 확인하고 싶다면 TIMESTAMPDIFF 함수를 사용
다음 쿼리는 MONTH를 입력해 개월 수를 반환함

Do it! TIMESTAMPDIFF 함수로 날짜 간의 개월 수 차 반환

```
SELECT TIMESTAMPDIFF(MONTH, '2023-12-31 23:59:59.9999999', '2023-01-01 00:00:00.0000000');
```

실행 결과

	TIMESTAMPDIFF(MONTH, '2023-12-31 23:59:59.9999999', '2023-01-01 00:00:00.0000000')
▶	-12

MONTH 대신 앞서 살펴본 시간 단위(YEAR, QUARTER 등)을 넣으면 원하는 단위로 날짜 간 차이를 확인할 수 있음

06-2 날짜 함수

▪ 지정한 날짜의 요일을 반환하는 함수 — DAYNAME

- DAYNAME 함수는 지정한 날짜의 요일을 반환

Do it! DAYNAME 함수로 특정 날짜의 요일 반환

```
SELECT DAYNAME('2023-08-20');
```

실행 결과

	DAYNAME('2023-08-20')
▶	Sunday

06-2 날짜 함수

- **날짜에서 연, 월, 주, 일을 값으로 가져오는 함수**
  — YEAR, MONTH, WEEK, DAY

  - 날짜 데이터의 일부만 사용하고 싶은 경우 유용하게 사용할 수 있는 함수임

Do it! YEAR, MONTH, WEEK, DAY 함수로 연, 월, 주, 일을 별도의 값으로 반환

```
SELECT
        YEAR('2023-08-20'),
        MONTH('2023-08-20'),
        WEEK('2023-08-20'),
        DAY('2023-08-20');
```

실행 결과

	YEAR('2023-08-20')	MONTH('2023-08-20')	WEEK('2023-08-20')	DAY('2023-08-20')
▶	2023	8	34	20

  - WEEK 함수의 경우 해당 날짜가 몇 번째 주에 해당하는지를 반환함

06-2 날짜 함수

▪ 날짜 형식을 변환하는 함수 — DATE_FORMAT, GET_FORMAT

1. DATE_FORMAT 함수는 날짜를 다양한 형식으로 표현해야 할 때 사용함.
나라마다 날짜를 표현하는 방식이 다르므로
날짜 형식으로 변환해야 할 때 DATE_FORMAT 함수가 필요함.

   다음 쿼리는 표준 날짜 형식을 미국에서 사용하는 날짜 형식으로 표시함.

Do it! DATE_FORMAT 함수로 날짜 형식 변경

```
SELECT DATE_FORMAT('2023-08-20 20:23:01', '%m/%d/%Y');
```

실행 결과

	DATE_FORMAT('2023-08-20 20:23:01', '%m/%d/%Y')
▶	08/20/2023

▪ 날짜 형식을 변환하는 함수 — DATE_FORMAT, GET_FORMAT

2. 앞선 쿼리에서 작성한 %m/%d/%Y 외에도 다양한 조합으로 자신이 원하는 날짜 형식으로 결과를 출력할 수 있음

다양한 날짜 형식 변환 예

```
SELECT DATE_FORMAT('2023-08-20 20:23:01', '%Y%m%d');
SELECT DATE_FORMAT('2023-08-20 20:23:01', '%Y.%m.%d');
SELECT DATE_FORMAT('2023-08-20 20:23:01', '%H:%i:%s');
```

06-2 날짜 함수

▪ 날짜 형식을 변환하는 함수 — DATE_FORMAT, GET_FORMAT

3. 일부 국가에서는 연, 월, 일 순서를 다르게 사용하는데,
우리가 각 국가나 지역별 표준 날짜 형식을 모두 알기는 어려움.
이러한 문제를 해결하기 위해 국가나 지역별 날짜 형식이 어떠한지 알기 위해
GET_FORMAT 함수를 사용할 수 있음

Do it! GET_FORMAT 함수로 국가나 지역별 날짜 형식 확인

```
SELECT GET_FORMAT(DATE, 'USA') AS USA,
GET_FORMAT(DATE, 'JIS') AS JIS,
GET_FORMAT(DATE, 'EUR') AS Europe,
GET_FORMAT(DATE, 'ISO') AS ISO,
GET_FORMAT(DATE, 'INTERNAL') AS INTERNAL;
```

실행 결과

	USA	JIS	Europe	ISO	INTERNAL
▶	%m.%d.%Y	%Y-%m-%d	%d.%m.%Y	%Y-%m-%d	%Y%m%d

▪ 날짜 형식을 변환하는 함수 — DATE_FORMAT, GET_FORMAT

4. DATE_FORMAT 함수와 GET_FORMAT 함수를 조합하여 현재 시간을 다양한 형식으로 표현할 수 있음.

Do it! DATE_FORMAT과 GET_FORMAT 함수 조합

```
SELECT DATE_FORMAT(NOW(), GET_FORMAT(DATE, 'USA')) AS USA,
DATE_FORMAT(NOW(), GET_FORMAT(DATE, 'JIS')) AS JIS,
DATE_FORMAT(NOW(), GET_FORMAT(DATE, 'EUR'))AS Europe,
DATE_FORMAT(NOW(), GET_FORMAT(DATE, 'ISO')) AS ISO,
DATE_FORMAT(NOW(), GET_FORMAT(DATE, 'INTERNAL')) AS INTERNAL;
```

실행 결과

	USA	JIS	Europe	ISO	INTERNAL
▶	08.20.2023	2023-08-20	20.08.2023	2023-08-20	20230820

06-3 집계 함수

▪ 조건에 맞는 데이터 개수를 세는 함수 — COUNT

1. 조건에 맞는 데이터의 개수를 세고 싶다면 COUNT 함수를 사용함

Do it! COUNT 함수로 데이터 개수 집계

```
SELECT COUNT(*) FROM customer;
```

실행 결과

	COUNT(*)
▶	599

2. stored_id 열 기준으로 그룹화하여
stored_id 열의 그룹마다 데이터가 몇 개씩 있는지 확인하는 쿼리

Do it! COUNT 함수와 GROUP BY 문 조합

```
SELECT store_id, COUNT(*) AS cnt FROM customer GROUP BY store_id;
```

실행 결과

	store_id	cnt
▶	1	326
	2	273

06-3 집계 함수

▪ 조건에 맞는 데이터 개수를 세는 함수 — COUNT

3. store_id 열과 active 열 기준으로 그룹화해서 store_id 열과 active 열 그룹마다 데이터가 각각 몇 개씩 있는지 확인하는 쿼리

Do it! COUNT 함수와 GROUP BY 문 조합: 열 2개 활용

```
SELECT store_id, active, COUNT(*) AS cnt FROM customer GROUP BY store_id, active;
```

실행 결과

	store_id	active	cnt
▶	1	1	318
	2	1	266
	2	0	7
	1	0	8

06-3 집계 함수

▪ 조건에 맞는 데이터 개수를 세는 함수 — COUNT

4. COUNT 함수를 사용할 때 주의할 점

집계할 때 해당 열에 있는 NULL값은 제외함.
그래서 전체 데이터 개수와 COUNT 함수로 얻은 데이터 개수가 다를 수 있음

Do it! NULL을 제외한 집계 확인

```
SELECT COUNT(*) AS all_cnt, COUNT(address2) AS ex_null FROM address;
```

실행 결과

	all_cnt	ex_null
▶	603	599

address2 열의 NULL은
COUNT 함수의 집계 대상이 아니므로 전체 행 개수와 다름

06-3 집계 함수

▪ 조건에 맞는 데이터 개수를 세는 함수 — COUNT

5. COUNT 함수를 사용할 때 DISTINCT 문을 조합하면
중복된 값을 제외한 데이터 개수를 집계할 수 있음

Do it! COUNT 함수와 DISTINCT 문 조합

```
SELECT COUNT(*), COUNT(store_id), COUNT(DISTINCT store_id) FROM customer;
```

실행 결과

	COUNT(*)	COUNT(store_id)	COUNT(DISTINCT store_id)
▶	599	599	2

store_id 열에는 1과 2 데이터가 중복되어 있음.
때문에 DISTINCT store_id를 통해
중복 데이터를 제외하여 조회한 뒤, COUNT 함수로 행 개수를 세면 2로 집계됨

06-3 집계 함수

▪ 데이터의 합을 구하는 함수 — SUM

1. payment 테이블에 있는 amount 열의 모든 데이터를 더하는 쿼리

Do it! SUM 함수로 amount 열의 데이터 합산

```
SELECT SUM(amount) FROM payment;
```

실행 결과

	SUM(amount)
▶	67406.56

2. SUM 함수와 GROUP BY 문을 조합하여 amount 열의 데이터를 합산하되 customer_id 열의 데이터를 그룹으로 나누어 합산하는 쿼리

Do it! SUM 함수와 GROUP BY 문 조합

```
SELECT customer_id, SUM(amount) FROM payment GROUP BY customer_id;
```

실행 결과

	customer_id	SUM(amount)
▶	1	118.68
	2	128.73
	3	135.74
	4	81.78
	5	144.62
	6	93.72
	7	151.67
	8	92.76
	9	89.77
	10	99.75

06-3 집계 함수

▪ 데이터의 합을 구하는 함수 — SUM

3. 만약 SUM 함수를 사용했을 때,
SUM 함수로 합산한 데이터가 기존 데이터 유형의 범위를 넘는 경우가 있음.

보통은 암시적 형 변환으로 오류가 발생하지는 않았지만
형 변환된 데이터 유형으로 결과가 반환된 것을 볼 수 있음.

다음 쿼리는 정수(int)로 된 데이터들을 더함.
col_1열의 데이터를 합산한 결과가 30억을 넘어 오버플로가 발생해야 정상임.

하지만 MySQL에서는 암시적 형 변환으로
DECIMAL 또는 BIGINT로 합산한 결과를 반환함

▪ 데이터의 합을 구하는 함수 — SUM

3. **Do it!** 암시적 형 변환으로 오버플로 없이 합산 결과를 반환

```
CREATE TABLE doit_overflow (
col_1 int,
col_2 int,
col_3 int
);

INSERT INTO doit_overflow VALUES (1000000000,1000000000, 1000000000);
INSERT INTO doit_overflow VALUES (1000000000,1000000000, 1000000000);
INSERT INTO doit_overflow VALUES (1000000000,1000000000, 1000000000);
SELECT SUM(col_1) FROM doit_overflow;
```

실행 결과

	SUM(col_1)
▶	3000000000

06-3 집계 함수

▪ 데이터의 평균을 구하는 함수 — AVG

1. payment 테이블에서 amount 열의 데이터 평균을 구하는 쿼리

Do it! AVG 함수로 amount 열의 데이터 평균 계산

```
SELECT AVG(amount) FROM payment;
```

실행 결과

	AVG(amount)
▶	4.201356

2. AVG 함수와 GROUP BY 문을 조합해 customer_id 열을 그룹화하고, 각 그룹별로 amount 열의 데이터 평균을 계산하는 쿼리

Do it! AVG 함수와 GROUP BY 문 조합

```
SELECT customer_id, AVG(amount) FROM payment GROUP BY customer_id;
```

실행 결과

	customer_id	AVG(amount)
▶	1	3.708750
	2	4.767778
	3	5.220769
	4	3.717273
	5	3.805789
	6	3.347143
	7	4.596061
	8	3.865000
	9	3.903043
	10	3.990000

06-3 집계 함수

▪ 최솟값 또는 최댓값을 구하는 함수 — MIN, MAX

1. payment 테이블에 있는 amount 열의 최솟값, 최댓값을 조회하는 쿼리

Do it! MIN과 MAX 함수로 amount 열의 최솟값과 최댓값 조회

```
SELECT MIN(amount), MAX(amount) FROM payment;
```

실행 결과

	MIN(amount)	MAX(amount)
▶	0.00	11.99

2. customer_id의 그룹별로 최솟값, 최댓값을 구한 쿼리임.

Do it! MIN과 MAX 함수 그리고 GROUP BY 문 조합

```
SELECT customer_id, MIN(amount), MAX(amount) FROM payment
GROUP BY customer_id;
```

실행 결과

	customer_id	MIN(amount)
▶	1	0.99
	2	0.99
	3	0.99
	4	0.99
	5	0.99
	6	0.99
	7	0.99
	8	0.99
	9	0.99
	10	0.99

06-3 집계 함수

▪ 부분합과 총합을 구하는 함수 — ROLLUP

- 데이터의 부분합과 총합을 구하고 싶다면
  GROUP BY 문과 ROLLUP 함수에 조합하면 됨
- ROLLUP 함수는 **GROUP BY (열 이름) WITH ROLLUP**에서 입력한 열을 기준으로 오른쪽에서 왼쪽으로 이동하면서 부분합과 총합을 구함

Do it! ROLLUP 함수로 부분합 계산

```
SELECT customer_id, staff_id, SUM(amount)
FROM payment
GROUP BY customer_id, staff_id WITH ROLLUP;
```

실행 결과

customer_id	staff_id	SUM(amount)
1	1	64.83
1	2	53.85
1	NULL	118.68
2	1	60.85
2	2	67.88
2	NULL	128.73
3	1	64.86
3	2	70.88
3	NULL	135.74
4	1	49.88

596	1	54.83
596	2	41.89
596	NULL	96.72
597	1	49.86
597	2	49.89
597	NULL	99.75
598	1	43.90
598	2	39.88
598	NULL	83.78
599	1	28.92
599	2	54.89
599	NULL	83.81
NULL	NULL	67406.56

06-3 집계 함수

▪ 데이터의 표준편차를 구하는 함수 — STDDEV, STDDEV_SAMP

- STDDEV 함수는 모든 값에 대한 표준편차를,
  STDDEV_SAMP 함수는 표본에 대한 표준편차를 구함.

Do it! STDDEV와 STDDEV_SAMP 함수로 표준편차 계산

```
SELECT STDDEV(amount), STDDEV_SAMP(amount) FROM payment;
```

실행 결과

	STDDEV(amount)	STDDEV_SAMP(amount)
▶	2.3628869202372846	2.3629605613923124

06-4 수학 함수

- 여기에서는 ABS, SIGN, ROUND, RAND 등의 여러 수학 함수를 다룸
- 수학 함수는 대부분 입력값과 같은 데이터 유형으로 값을 반환하지만 LOG, EXP, SQRT 같은 함수는 입력값을 실수형인 float으로 자동 변환하고 반환함

▪ 절댓값을 구하는 함수 — ABS

1.

Do it! ABS 함수에 입력한 숫자를 절댓값으로 반환

```
SELECT ABS(-1.0), ABS(0.0), ABS(1.0);
```

실행 결과

	ABS(-1.0)	ABS(0.0)	ABS(1.0)
▶	1.0	0.0	1.0

06-4 수학 함수

▪ 절댓값을 구하는 함수 — ABS

2. 수식을 입력해 수식을 계산한 결과를 절댓값으로 반환하는 쿼리

Do it! ABS 함수에 입력한 수식의 결과를 절댓값으로 반환

```
SELECT a.amount - b.amount AS amount, ABS(a.amount - b.amount) AS abs_amount
FROM payment AS a
        INNER JOIN payment AS b ON a.payment_id = b.payment_id-1;
```

payment 테이블에서 payment_id가 현재보다 -1 값인 paymeny_id를 찾아서 amount값을 뺀 결과임.

실행 결과

	amount	abs_amount
▶	2.00	2.00
	-5.00	5.00
	5.00	5.00
	-9.00	9.00
	5.00	5.00
	0.00	0.00

▪ 절댓값을 구하는 함수 — ABS

3. 자료형의 범위를 넘으면 오버플로가 발생 -> MySQL 서버에서는 BIGINT로 암시적 형 변환이 되어 값이 반환됨.

Do it! 암시적 형 변환으로 오버플로 없이 절댓값을 반환

```
SELECT ABS(-2147483648);
```

실행 결과

	ABS(-2147483648)
▶	2147483648

[Field Types]를 클릭해 보면 데이터 유형을 확인할 수 있는데 자동으로 BIGINT로 형 변환된 것을 확인할 수 있음

#	Field	Schema	Table	Type	Character Set	Display Size	Precision	Scale
1	ABS(-2147483648)			BIGINT	binary	11	10	0

06-4 수학 함수

▪ 양수 또는 음수 여부를 판단하는 함수 — SIGN

- SIGN 함수는 지정한 값이나 수식 결과값이 양수, 음수, 0인지를 판단하여 각각 1, -1, 0을 반환함

1.

Do it! SIGN 함수로 입력한 숫자가 양수, 음수, 0인지를 판단

```
SELECT SIGN(-256), SIGN(0), SIGN(256);
```

실행 결과

	SIGN(-256)	SIGN(0)	SIGN(256)
▶	-1	0	1

2.

Do it! SIGN 함수로 수식의 결과가 양수, 음수, 0인지를 판단

```
SELECT a.amount - b.amount AS amount, SIGN(a.amount - b.amount) AS abs_amount
FROM payment AS a
        INNER JOIN payment AS b ON a.payment_id = b.payment_id-1;
```

실행 결과

	amount	abs_amount
▶	2.00	1
	-5.00	-1
	5.00	1
	-9.00	-1
	5.00	1

06-4 수학 함수

▪ 천장값과 바닥값을 구하는 함수 — CEILING, FLOOR

- CEILING 함수는 천장값 - 입력한 숫자보다 크거나 같은 최소 정수
  예) 2.4는 3을 반환함
- FLOOR 함수는 바닥값 - 지정한 숫자보다 작거나 같은 최대 정수
  예) 2.4는 2를 반환함

Do it! CEILING 함수로 천장값 반환

```
SELECT CEILING(2.4), CEILING(-2.4), CEILING(0.0);
```

실행 결과

	CEILING(2.4)	CEILING(-2.4)	CEILING(0.0)
▶	3	-2	0

Do it! FLOOR 함수로 바닥값 반환

```
SELECT FLOOR(2.4), FLOOR(-2.4), FLOOR(0.0);
```

실행 결과

	FLOOR(2.4)	FLOOR(-2.4)	FLOOR(0.0)
▶	2	-3	0

06-4 수학 함수

▪ 반올림을 반환하는 함수 — ROUND

ROUND 함수의 기본 형식

ROUND(숫자, 자릿수)

- 첫 번째 인수에는 반올림할 대상인 숫자값 또는 열 이름을 넣어 사용함
- 두 번째 인수에는 반올림하여 표현할 자릿수를 입력함

- 예를 들어 첫 번째 인수는 123.1234이고,
  두 번째 인수가 3이라면 ROUND 함수는 123.123을 반환함

- 자릿수에 또한 양수나 음수를 지정할 수 있는데
  양수로는 소수부터 반올림하고, 음수로는 정수부터 반올림함

06-4 수학 함수

▪ 반올림을 반환하는 함수 — ROUND

1. **Do it! ROUND 함수로 소수점 셋째 자리까지 반올림**

```
SELECT ROUND(99.9994, 3), ROUND(99.9995, 3);
```

실행 결과

	ROUND(99.9994, 3)	ROUND(99.9995, 3)
▶	99.999	100.000

2. **Do it! ROUND 함수로 소수와 정수를 따로 반올림**

```
SELECT ROUND(234.4545, 2), ROUND(234.4545, -2);
```

실행 결과

	ROUND(234.4545, 2)	ROUND(234.4545, -2)
▶	234.45	200

3. **Do it! 정수 부분의 길이보다 큰 자릿수를 입력한 경우**

```
SELECT ROUND(748.58, -1);
SELECT ROUND(748.58, -2);
SELECT ROUND(748.58, -4);
```

실행 결과

	ROUND(748.58, -1)
▶	750

	ROUND(748.58, -2)
▶	700

	ROUND(748.58, -4)
▶	0

06-4 수학 함수

▪ 로그를 구하는 함수 — LOG

- 로그 함수는 지수 함수의 역함수이므로 다음과 같은 관계를 나타냄

$$\log_b(a)=c \leftrightarrow b^c=a$$

LOG 함수의 기본 형식

LOG(로그 계산할 숫자 또는 식, 밑)

밑을 생략한 경우에는 자연 로그가 되며 이때 기본값으로 e가 설정됨(2.718281828...)

Do it! LOG 함수로 로그 10을 계산

```
SELECT LOG(10);
```

실행 결과

	LOG(10)
▶	2.302585092994046

Do it! LOG 함수로 로그 10의 5를 계산

```
SELECT LOG(10, 5);
```

실행 결과

	LOG(10, 5)
▶	0.6989700043360187

06-4 수학 함수

▪ e의 n 제곱값을 구하는 함수 — EXP

- e의 n 제곱값

EXP 함수 기본 형태

EXP(지수 계산할 숫자 또는 식)

Do it! EXP 함수로 지수 1.0을 계산

```
SELECT EXP(1.0);
```

실행 결과

	EXP(1.0)
▶	2.718281828459045

Do it! EXP 함수로 지수 10을 계산

```
SELECT EXP(10);
```

실행 결과

	EXP(10)
▶	22026.465794806718

Do it! LOG 함수와 EXP 함수로 결과 확인

```
SELECT EXP(LOG(20)), LOG(EXP(20));
```

실행 결과

	EXP(LOG(20))	LOG(EXP(20))
▶	19.999999999999996	20

06-4 수학 함수

▪ 거듭제곱값을 구하는 함수 — POWER

POWER 함수의 기본 형식

POWER(숫자 또는 수식, 지수)

Do it! POWER 함수로 거듭제곱 계산

```
SELECT POWER(2,3), POWER(2,10), POWER(2.0, 3);
```

실행 결과

	POWER(2,3)	POWER(2,10)	POWER(2.0, 3)
▶	8	1024	8

06-4 수학 함수

▪ 제곱근을 구하는 함수 — SQRT

SQRT 함수의 기본 형식

SQRT(숫자 또는 수식)

Do it! SQRT 함수로 제곱근 계산

```
SELECT SQRT(1.00), SQRT(10.00);
```

실행 결과

	SQRT(1.00)	SQRT(10.00)
▶	1	3.1622776601683795

06-4 수학 함수

▪ 난수를 구하는 함수 — RAND

- RAND 함수는 0부터 1 사이의 실수형 난수를 반환함

RAND 함수의 기본 형식

RAND(인자)

- 만약 인자를 지정하지 않으면 임의로 초깃값을 설정함
- 같은 인자를 설정하면 RAND 함수는 같은 결과를 반환함

Do it! RAND 함수로 난수 계산

```
SELECT RAND(100), RAND(), RAND();
```

실행 결과

	RAND(100)	RAND()	RAND()
▶	0.17353134804734155	0.0726371417498562	0.9080051518119919

06-4 수학 함수

▪ 난수를 구하는 함수 — RAND

2. **Do it! 인수가 없는 RAND 함수로 난수 계산**

```
DELIMITER $$
CREATE PROCEDURE rnd()
BEGIN
DECLARE counter INT;
SET counter = 1;

WHILE counter < 5 DO
        SELECT RAND() Random_Number;
        SET counter = counter + 1;
END WHILE;
END $$
DELIMITER ;

call rnd();
```

실행 결과

	Random_Number
▶	0.9633106123314338

06-4 수학 함수

▪ 삼각 함수 — COS, SIN, TAN, ATAN

Do it! COS 함수 계산

```
SELECT COS(14.78);
```

실행 결과

	COS(14.78)
▶	-0.5994654261946543

Do it! SIN 함수 계산

```
SELECT SIN(45.175643);
```

실행 결과

	SIN(45.175643)
▶	0.929607286611012

Do it! TAN 함수 계산

```
SELECT TAN(PI()/2), TAN(.45)
```

실행 결과

	TAN(PI()/2)	TAN(.45)
▶	1.633123935319537e16	0.4830550656165784

▪ 삼각 함수 — COS, SIN, TAN, ATAN

2.

Do it! ATAN 함수 계산

```
SELECT ATAN(45.87) AS atanCalc1,
       ATAN(-181.01) AS atanCalc2,
       ATAN(0) AS atanCalc3,
       ATAN(0.1472738) AS atanCalc4,
       ATAN(197.1099392) AS atanCalc5;
```

실행 결과

	atanCalc1	atanCalc2	atanCalc3	atanCalc4	atanCalc5
▶	1.548999038348081	-1.5652718263440326	0	0.1462226769376524	1.5657230594360985

06-5 순위 함수

▪ 유일한 값으로 순위를 부여하는 함수 — ROW_NUMBER

- 모든 행에 유일한 값으로 순위를 부여. 실행한 결과에는 같은 순위가 없음

ROW_NUMBER 함수의 기본 형식

```
ROW_NUMBER() OVER([PARTITION BY 열] ORDER BY 열)
```

- 데이터 그룹별 순위를 부여하고 싶다면 **PARTITION BY** 열 옵션을 사용함

06-5 순위 함수

▪ 유일한 값으로 순위를 부여하는 함수 — ROW_NUMBER

1. customer_id 열을 그룹화한 뒤,
   그룹별로 amount의 값을 합한 결과를 내림차순 정렬한 결과에 따라
   ROW_NUMBER 함수로 순위를 부여하는 쿼리

Do it! ROW_NUMBER 함수로 순위 부여

```
SELECT ROW_NUMBER() OVER(ORDER BY amount DESC) AS num, customer_id, amount
FROM (
        SELECT customer_id, SUM(amount) AS amount
        FROM payment GROUP BY customer_id
) AS x;
```

실행 결과

num	customer_id	amount
1	526	221.55
2	148	216.54
3	144	195.58
4	137	194.61
5	178	194.61
6	459	186.62
7	469	177.60
8	468	175.61
9	236	175.58
10	181	174.66

동일한 amount값에도 순위가 다른 점도 확인(num열의 4, 5위)

▪ 유일한 값으로 순위를 부여하는 함수 — ROW_NUMBER

2. 앞선 쿼리가 거의 동일하나
ORDER BY 문에 customer_id 열을 내림차순으로 정렬하는 조건을 추가

Do it! 내림차순 정렬한 결과에 ROW_NUMBER 함수로 순위 부여

```
SELECT ROW_NUMBER() OVER(ORDER BY amount DESC, customer_id DESC)
AS num, customer_id, amount
FROM (
        SELECT customer_id, SUM(amount) AS amount
        FROM payment GROUP BY customer_id
) AS x;
```

실행 결과

num	customer_id	amount
1	526	221.55
2	148	216.54
3	144	195.58
4	178	194.61
5	137	194.61
6	459	186.62
7	469	177.60
8	468	175.61
9	236	175.58
10	181	174.66

06-5 순위 함수

▪ 유일한 값으로 순위를 부여하는 함수 — ROW_NUMBER

3. 전체 데이터가 아니라 그룹별로 순위를 부여해야 할 때가 있음.

   예를 들어 전교생 대상으로 석차를 구할 때와
   학년별 석차를 구할 때를 생각하면 됨.

   그룹별 순위를 부여하려면 PARTITION 문을 사용해야 함.

06-5 순위 함수

▪ 유일한 값으로 순위를 부여하는 함수 — ROW_NUMBER

3. staff_id 열을 그룹화하기 위해 PARTITION BY staff_id를 추가

Do it! PARTITION BY 문으로 사용해 그룹별 순위 부여

```
SELECT staff_id,
ROW_NUMBER() OVER(PARTITION BY staff_id ORDER BY amount DESC, customer_id ASC)
AS num, customer_id, amount
FROM (
        SELECT customer_id, staff_id, SUM(amount) AS amount
        FROM payment GROUP BY customer_id, staff_id
) AS x;
```

실행 결과

staff_id	num	customer_id	amount
1	1	176	126.74
1	2	137	115.77
1	3	526	111.79
1	4	178	108.81
1	5	459	108.78
1	596	281	15.95
1	597	248	15.93
1	598	344	12.95
1	599	320	10.95
2	1	187	110.81
2	2	148	110.78
2	3	526	109.76
2	4	522	108.79
2	5	211	108.77
2	6	309	104.81

결과를 보면 staff_id 열이 1과 2 그룹으로 묶였는데,
각 그룹마다 순위가 부여되었음을 확인할 수 있음

06-5 순위 함수

▪ 우선순위를 고려하지 않고 순위를 부여하는 함수 — RANK

- RANK 함수는 ROW_NUMBER 함수와 비슷하지만
  같은 순위를 처리하는 방법은 다름
- RANK 함수를 활용할 때에는 우선순위를 따지지 않고 같은 순위를 부여함

RANK 함수의 기본 형식

```
RANK() OVER([PARTITION BY 열] ORDER BY 열)
```

Do it! RANK 함수로 순위 부여

```
SELECT RANK() OVER(ORDER BY amount DESC) AS num, customer_id, amount
FROM (
        SELECT customer_id, SUM(amount) AS amount
        FROM payment GROUP BY customer_id
) AS x;
```

실행 결과

	num	customer_id	amount
▶	1	526	221.55
	2	148	216.54
	3	144	195.58
	4	137	194.61
	4	178	194.61
	6	459	186.62
	7	469	177.60
	8	468	175.61
	9	236	175.58
	10	181	174.66

06-5 순위 함수

▪ 건너뛰지 않고 순위를 부여하는 함수 — DENSE_RANK

- DENSE_RANK 함수는 RANK 함수와 동일하지만 RANK 함수와 달리 같은 순위에 있는 데이터 개수를 무시하고 순위를 매긴다는 점은 다름
- 예를 들어 1위가 3개여도 그다음 데이터는 4위가 아닌 2위가 됨

DENSE_RANK 함수의 기본 형식

```
DENSE_RANK() OVER([PARTITION BY 열] ORDER BY 열)
```

Do it! DENSE_RANK 함수로 순위 부여

```
SELECT DENSE_RANK() OVER(ORDER BY amount DESC) AS num,
customer_id, amount
FROM (
        SELECT customer_id, SUM(amount) AS amount
        FROM payment GROUP BY customer_id
) AS x;
```

실행 결과

num	customer_id	amount
1	526	221.55
2	148	216.54
3	144	195.58
4	137	194.61
4	178	194.61
5	459	186.62
6	469	177.60
7	468	175.61
8	236	175.58
9	181	174.66

06-5 순위 함수

▪ 그룹 순위를 부여하는 함수 — NTILE

- 인자로 지정한 개수만큼 데이터 행을 그룹화한 다음 각 그룹에 순위를 부여함
- 각 그룹은 1부터 순위가 매겨지며, 이때 순위는 각 행의 순위가 아니라 행이 속한 그룹의 순위임

NTILE 함수의 기본 형식

NTILE(정수) OVER([PARTITION BY 열] ORDER BY 열)

Do it! 내림차순으로 정렬한 결과에 NTILE 함수로 순위 부여

```
SELECT NTILE(100) OVER(ORDER BY amount DESC) AS num, customer_id, amount
FROM (
        SELECT customer_id, SUM(amount) AS amount
        FROM payment GROUP BY customer_id
) AS x;
```

실행 결과

	num	customer_id	amount
▶	1	526	221.55
	1	148	216.54
	1	144	195.58
	1	137	194.61
	1	178	194.61
	1	459	186.62
	2	469	177.60
	2	468	175.61
	2	236	175.58
	2	181	174.66

전체 행을 100개의 그룹으로 쪼갠 뒤, 각 행마다 속한 그룹의 순위를 부여

06-6 분석 함수

- 분석 함수는 데이터 그룹을 기반으로 앞뒤 행을 계산하거나
  그룹에 대한 누적 분포, 상대 순위 등을 계산함

▪ 앞 또는 뒤 행을 참조하는 함수 — LAG, LEAD

- 앞서 하루 전 날짜 데이터와 오늘 날짜 데이터를 비교할 때
  SELF JOIN 문을 전일 대비 오늘의 수익을 구할 수 있었음
- 이렇게 앞뒤 행을 비교하여 데이터 처리를 해야 하는 경우
  LAG 또는 LEAD 함수를 사용하면 됨
- 이 함수들을 사용하면 SELF JOIN 문을 따로 작성하지 않아도 돼 간편함

06-6 분석 함수

▪ 앞 또는 뒤 행을 참조하는 함수 — LAG, LEAD

- LAG 함수는 현재 행에서 바로 앞의 행을,
  LEAD 함수는 현재 행에서 바로 뒤의 행을 조회해 데이터를 처리함
- 인자에 전달한 값에 따라 이전 또는 이후
  몇 번째 행(데이터)를 참조할지 결정할 수도 있음

LAG와 LEAD 함수의 기본 형식

```
LAG(또는 LEAD)(열 이름, 참조 위치)
OVER([PARTITION BY 열] ORDER BY 열)
```

06-6 분석 함수

▪ 앞 또는 뒤 행을 참조하는 함수 — LAG, LEAD

1. payment 테이블에서 현재 행 기준으로 앞 또는 뒤의 행을 참조.

Do it! LAG와 LEAD 함수로 앞뒤 행 참조

```
SELECT x.payment_date,
            LAG(x.amount) OVER(ORDER BY x.payment_date ASC) AS lag_amount, amount,
            LEAD(x.amount) OVER(ORDER BY x.payment_date ASC) AS lead_amount
FROM (
            SELECT date_format(payment_date, '%y-%m-%d') AS payment_date, SUM(amount) AS amount
            FROM payment GROUP BY date_format(payment_date, '%y-%m-%d')
) AS x
ORDER BY x.payment_date;
```

실행 결과

payment_date	lag_amount	amount	lead_amount
05-05-24	NULL	29.92	573.63
05-05-25	29.92	573.63	754.26
05-05-26	573.63	754.26	684.34
05-05-27	754.26	684.34	804.04
05-05-28	684.34	804.04	648.46
05-05-29	804.04	648.46	628.42
05-05-30	648.46	628.42	700.37
05-05-31	628.42	700.37	57.84
05-06-14	700.37	57.84	1376.52
05-06-15	57.84	1376.52	1349.76

▪ 앞 또는 뒤 행을 참조하는 함수 — LAG, LEAD

2. LAG와 LEAD 함수의 인자로 2를 설정해 2행 앞이나 뒤를 참조하도록 작성

Do it! LAG와 LEAD 함수로 2칸씩 앞뒤 행 참조

```
SELECT x.payment_date,
            LAG(x.amount, 2) OVER(ORDER BY x.payment_date ASC) AS lag_amount, amount,
            LEAD(x.amount, 2) OVER(ORDER BY x.payment_date ASC) AS lead_amount
FROM (
            SELECT date_format(payment_date, '%y-%m-%d') AS payment_date, SUM(amount) AS amount
            FROM payment GROUP BY date_format(payment_date, '%y-%m-%d')
) AS x
ORDER BY x.payment_date;
```

실행 결과

payment_date	lag_amount	amount	lead_amount
05-05-24	NULL	29.92	754.26
05-05-25	NULL	573.63	684.34
05-05-26	29.92	754.26	804.04
05-05-27	573.63	684.34	648.46
05-05-28	754.26	804.04	628.42
05-05-29	684.34	648.46	700.37
05-05-30	804.04	628.42	57.84
05-05-31	648.46	700.37	1376.52
05-06-14	628.42	57.84	1349.76
05-06-15	700.37	1376.52	1332.75

06-6 분석 함수

▪ 누적 분포를 계산하는 함수 — CUME_DIST

- 그룹에서 데이터 값이 포함되는 위치의 누적 분포를 계산

CUME_DIST 함수의 기본 형식

CUME_DIST() OVER([PARTITION BY 열] ORDER BY 열)

Do it! CUME_DIST 함수로 누적 분폿값 계산

```
SELECT x.customer_id, x.amount, CUME_DIST() OVER(ORDER BY x.amount DESC)
FROM (
        SELECT customer_id, sum(amount) AS amount
        FROM payment GROUP BY customer_id
) AS X
ORDER BY x.amount DESC;
```

실행 결과

customer_id	amount	CUME_DIST() OVER (ORDER BY x.amount DESC)
526	221.55	0.0016694490818030 05
148	216.54	0.0033388981636060 1
144	195.58	0.0050083472454090 15
137	194.61	0.0083472454090150 25
178	194.61	0.0083472454090150 25
459	186.62	0.0100166944908180 3
469	177.60	0.0116861435726210 35
468	175.61	0.0133555926544240 4
236	175.58	0.0150250417362270 46
181	174.66	0.0166944908180300 5
176	173.63	0.0183639398998330 55
50	169.65	0.0200333889816360 6

06-6 분석 함수

▪ 상대 순위를 계산하는 함수 — PERCENT_RANK

- 지정한 그룹 또는 쿼리 결과로 이루어진 그룹 내의 상대 순위
- CUME_DIST 함수와 유사하지만 누적 분포가 아닌 분포 순위라는 점이 다름

PERCENT_RANK 함수의 기본 형식

PERCENT_RANK() OVER([PARTITION BY 열] ORDER BY 열)

Do it! PERCENT_RANK 함수로 상위 분포 순위를 계산

```
SELECT x.customer_id, x.amount, PERCENT_RANK() OVER (ORDER BY x.amount DESC)
FROM (
        SELECT customer_id, sum(amount) AS amount
        FROM payment GROUP BY customer_id
) AS X
ORDER BY x.amount DESC;
```

실행 결과

customer_id	amount	PERCENT_RANK() OVER (ORDER BY x.amount DESC)
526	221.55	0
148	216.54	0.0016722408026755853
144	195.58	0.0033444816053511705
137	194.61	0.005016722408026756
178	194.61	0.005016722408026756
459	186.62	0.0083612040133779926
469	177.60	0.010033444816053512
468	175.61	0.011705685618729096
236	175.58	0.013377926421404682
181	174.66	0.015050167224080268

어떤 고객이 전체 고객 중 상위 몇 퍼센트에 위치하는지를 확인

▪ (추가) CUME_DIST와 PERCENT_RANK 비교

- CUME_DIST : 주어진 그룹에 대한 상대적인 누적분포도 값을 반환
- PERCENT_RANK : 그룹 안에서 해당 로우의 값보다 작은 값의 비율

전체 개수(10)에서
자기 순위까지 누적된
자료 개수(5)의 비율

= 5/10 = 0.5

= 7/10 = 0.7

순위	row_num	cume_dist	percent_rank
1	1	0.1	0
2	2	0.2	0.1111111111111111
3	3	0.5	0.2222222222222222
3	4	0.5	0.2222222222222222
3	5	0.5	0.2222222222222222
4	6	0.7	0.5555555555555556
4	7	0.7	0.5555555555555556
5	8	0.8	0.7777777777777778
6	9	1	0.8888888888888888
6	10	1	0.8888888888888888

자신을 제외한 전체 개수(9)에서
자기 순위 앞에 있는
자료 개수(2)의 비율

= 2/9 = 0.2222222222222

= 7/9 = 0.7777777777778

06-6 분석 함수

▪ 첫 행 또는 마지막 행의 값을 구하는 함수 — FIRST_VALUE, LAST_VALUE

- 그룹에서 첫 행 또는 마지막 행의 값을 구하여 데이터를 비교할 때 많이 사용함
- FIRST_VALUE 함수는 그룹에서 정렬된 데이터에서 첫 번째 행의 값을,
  LAST_VALUE 함수는 마지막 행의 값을 반환함

FIRST_VALUE와 LAST_VALUE의 기본 형식

```
FIRST_VALUE(열) OVER ([PARTITION BY] ORDER BY 열)
LAST_VALUE(열) OVER ([PARTITION BY] ORDER BY 열)
```

06-6 분석 함수

- **첫 행 또는 마지막 행의 값을 구하는 함수 — FIRST_VALUE, LAST_VALUE**
  - payment 테이블에서 각 일자별로 amount 합계를 구한 다음 payment_date 열을 기준으로 오름차순 정렬하여,
    FIRST_VALUE 함수를 통해 가장 낮은 일자에 대한 amount 값을 구하고,
    LAST VALUE 함수를 사용하여 가장 높은 일자에 대한 amount 값을 구함
  - 그리고 가장 FIRST_VALUE 값과 현재 값의 차이가 얼마인지를 조회하는 쿼리

Do it! FIRST_VALUE 함수로 가장 높은 값 조회

```
SELECT x.payment_date, x.amount,
                FIRST_VALUE(x.amount) OVER(ORDER BY x.payment_date) AS f_value,
                LAST_VALUE(x.amount) OVER(ORDER BY x.payment_date RANGE BETWEEN UNBOUNDED
PRECEDING AND UNBOUNDED FOLLOWING) AS l_value,
                x.amount - FIRST_VALUE(x.amount) OVER(ORDER BY x.payment_date) AS increase_
amount
FROM (
                SELECT date_format(payment_date, '%y-%m-%d') AS payment_date,
                SUM(amount) AS amount
                FROM payment GROUP BY date_format(payment_date, '%y-%m-%d')
) AS x
ORDER BY x.payment_date;
```

실행 결과

payment_date	amount	f_value	l_value	increase_amount
05-05-24	29.92	29.92	514.18	0.00
05-05-25	573.63	29.92	514.18	543.71
05-05-26	754.26	29.92	514.18	724.34
05-05-27	684.34	29.92	514.18	654.42
05-05-28	804.04	29.92	514.18	774.12
05-05-29	648.46	29.92	514.18	618.54
05-05-30	628.42	29.92	514.18	598.50
05-05-31	700.37	29.92	514.18	670.45
05-06-14	57.84	29.92	514.18	27.92
05-06-15	1376.52	29.92	514.18	1346.60