

01번: 내장 함수 결과 쓰기

함수	결과
ABS(-15)	15
CEIL(15.7)	16
COS(3.14159)	≈ -1
FLOOR(15.7)	15
LOG(10, 100)	2
MOD(11, 4)	3
POWER(3, 2)	9
ROUND(15.7)	16
SIGN(-15)	-1
TRUNCATE(15.7, 0)	15
CHAR(67 USING utf8)	'C'
CONCAT('HAPPY', 'Birthday')	'HAPPYBirthday'
LOWER('Birthday')	'birthday'
LPAD('Page 1', 15, '*')	'*****Page 1'
REPLACE('JACK', 'J', 'BL')	'BLACK'
RPAD('Page 1', 15, '*')	'Page 1*****'
SUBSTR('ABCDEFGH', 3, 4)	'CDEF'
TRIM(LEADING 0 FROM '00AA00')	'AA00'
UPPER('Birthday')	'BIRTHDAY'
ASCII('A')	65

LENGTH('Birthday')	8
ADDDATE('2024-02-14', INTERVAL 10 DAY)	'2024-02-24'
LAST_DAY(SYSDATE())	현재 날짜 기준 해당 달의 마지막 날짜 (예: 2025-06-30)
NOW()	현재 날짜 및 시간 (예: 2025-06-20 12:34:56)
DATE_FORMAT(SYSDATE(), '%Y')	'2025'
CONCAT(123)	'123'
STR_TO_DATE('12 05 2024', '%d %m %Y')	'2024-05-12'
CAST('12.3' AS DECIMAL(3,1))	12.3
IF(1=1, 'aa', 'bb')	'aa'
IFNULL(123, 345)	123
IFNULL(NULL, 123)	123

2-1

bookid	price
1	10000
2	20000
3	NULL

2-2

bookid	IFNULL(price, 0)
1	10000
2	20000
3	0

2-3

bookid	price
3	NULL

2-4

결과: 0행

2-5

bookid	price+100
1	10100
2	20100
3	NULL

2-6

SUM(price) = NULL
AVG(price) = NULL
COUNT(*) = 0

2-7

3 | 2

2-8

30000 | 15000

2-9

03번: NULL 포함된 테이블 R

sql

복사편집

```
INSERT INTO R VALUES(NULL, 10);  
INSERT INTO R VALUES(12, NULL);  
INSERT INTO R VALUES(NULL, NULL);  
INSERT INTO R VALUES(10, 12);
```

1)

sql

```
SELECT COUNT(A) FROM R;
```

- A 값이 **NULL** 아닌 것만 카운트 → 2개 (12, 10)

결과: 2

(2)

sql

```
SELECT * FROM R WHERE A IN (10, 12, NULL);
```

- **NULL**과의 비교는 항상 **false** → **A IN (10, 12)**만 평가됨

결과:

A	B
12	NULL
10	12

(3)

sql

```
SELECT A, COUNT(*) FROM R GROUP BY A;
```

- 그룹핑 시 **NULL**도 별도 그룹으로 간주

A	COUNT
NULL	2

10 1

12 1

04번

(1) **SELECT bookid, bookname, price FROM Book;**

→ 📌 = 전체 출력

(2) **SELECT bookid, bookname, price FROM Book LIMIT 5;**

→ 상위 5개 행 출력

(3) **SELECT bookid, bookname, price FROM Book ORDER BY price LIMIT 5;**

→ 가격이 낮은 순으로 정렬한 뒤 5개 출력

(4)

sql

SET @RNUM := 0;

**SELECT bookid, bookname, price, @RNUM := @RNUM + 1 AS ROWNUM
FROM Book**

WHERE @RNUM < 5;

-> 아무것도 안 나옴

(5)

sql

```
SELECT bookid, bookname, price, @RNUM := @RNUM + 1 AS ROWNUM
FROM Book, (SELECT @RNUM := 0) tmp
WHERE @RNUM < 5;
```

4번과 같음

(6)

sql

```
SELECT bookid, bookname, price, @RNUM := @RNUM + 1 AS ROWNUM
FROM (
    SELECT * FROM Book ORDER BY price
) AS b, (SELECT @RNUM := 0) AS init
WHERE @RNUM < 5;
```

가격 기준 상위 4개 출력됨

(7)

sql

```
SELECT bookid, bookname, price, @RNUM := @RNUM + 1 AS ROWNUM
FROM (
    SELECT * FROM Book ORDER BY price
) AS b, (SELECT @RNUM := 0) AS init
LIMIT 5;
```

정렬된 결과 중 상위 5개를 출력하고, 거기에 순번 붙이기 가능

05번. 부속질의 (Subquery)에 대한 설명

(1)

sql

복사편집

```
SELECT custid, (SELECT address
```

```

        FROM Customer cs
        WHERE cs.custid = od.custid) 'address',
        SUM(saleprice) 'total'
FROM Orders od
GROUP BY od.custid;

```

해석

- **Orders** 테이블에서 **custid**별로 **SUM(saleprice)** 구함
- 동시에 해당 **custid**의 주소를 **Customer**에서 서브쿼리로 가져옴

✓ 의미: 고객별 총 구매금액과 주소를 조회

✓ 서브쿼리는 **SELECT**절에 있는 스칼라 서브쿼리

(2)

sql

복사편집

```

SELECT cs.name, s
FROM (
    SELECT custid, AVG(saleprice) s
    FROM Orders
    GROUP BY custid
) s,
Customer cs
WHERE cs.custid = s.custid;

```

해석

- 서브쿼리: 고객별 평균 구매금액 구함
- 메인쿼리: **Customer** 테이블과 조인하여 이름과 평균 금액 출력

✓ 의미: 고객 이름과 평균 구매 금액 조회


✓ 인라인 뷰 + **JOIN**

(3)

sql

복사편집

```
SELECT SUM(saleprice) 'total'
FROM Orders od
WHERE EXISTS (
    SELECT *
    FROM Customer cs
    WHERE cs.custid = od.custid
    AND cs.grade = 3
    AND cs.custid = od.custid
);
```

 해석

- **Orders**의 각 행마다 조건을 만족하는 **Customer**가 존재하는지 확인
- 조건: **grade = 3**이면서 **custid** 일치

✓ 의미: 등급이 3인 고객의 구매 금액 합계

✓ **EXISTS**는 조건 만족하는 행이 존재하기만 하면 OK

06번.

(1)

sql

```
SELECT deptno FROM Dept
WHERE deptno NOT IN (SELECT deptno FROM Emp);
```

Emp에 **NULL**이 있으면 전체 결과가 **NULL**로 무시됨

(2)

sql

```
SELECT deptno FROM Dept A
WHERE NOT EXISTS (
```



```
SELECT * FROM Emp B WHERE A.deptno = B.deptno
);
```

Emp에 해당 부서 배치된 사원이 없을 때만 TRUE

(3)

sql

```
SELECT B.deptno
FROM EMP A RIGHT OUTER JOIN Dept B ON A.deptno = B.deptno
WHERE empno IS NULL;
```

사원이 없는 부서

(4)

sql

```
SELECT deptno FROM Dept
WHERE deptno != ANY (SELECT deptno FROM EMP);
```

- 모든 값과 달라야만 조건 만족
- 한 명이라도 배정된 사원이 있으면 해당 부서 제외 안됨

07번.

(1) 부속질의 사용

sql

```
SELECT name
FROM Customer
WHERE address LIKE '%대한민국%'
```

```
AND name NOT IN (  
    SELECT name FROM Customer  
    WHERE custid IN (SELECT custid FROM Orders)  
);
```

- 이름이 중복될 수 있고, **NOT IN**은 **NULL** 문제 발생 가능성 있음

(2) EXISTS 사용

```
sql  
SELECT name  
FROM Customer c1  
WHERE address LIKE '%대한민국%'  
    AND NOT EXISTS (  
        SELECT name FROM Customer  
        WHERE custid IN (SELECT custid FROM Orders)  
        AND c1.name = name  
    );
```

→ 주소가 대한민국이고, 주문기록과 연결되는 이름이 존재하지 않는 경우

(3) 조인 사용

```
sql  
SELECT c1.name  
FROM Customer c1, Customer c2  
WHERE c1.name = c2.name  
    AND c1.address LIKE '%대한민국%'
```

고객 이름만 같다고 판단 → **custid**로 매칭하는 게 아님
→ 정확하지 않음 (동명이인 가능성)

08번. 테이블 **R**, **S**에 대해 주어진 **SQL** 결과 예측

결과 **GROUP BY:** (회원번호, 등급)

회원번호 등급 합계(SUM)

1 1 60000

1 2 3000

2 1 40000

09번. SQL 실행 순서 번호로 쓰기

① → ② → ③ → ④ → ⑤ → ⑥

10번. 뷰(view) 정의 및 결과 예측

테이블 **R(A, B)**

A	B
a	1000
a	2000
b	1000
NULL	3000

→ 뷰 **V**는 조건에 따라 다음 **3**행 포함:

A	B
a	1000
a	2000
NULL	3000

→ **R** 전체에서 **B**가 **2000** 이상인 값: **2000, 3000** → 합계: **5000**

결과: 5000

11번. 뷰(view)의 장점과 개념 정리

뷰의 장점

1. 보안성 향상: 테이블 전체가 아닌 제한된 컬럼만 보여줄 수 있음
 2. 복잡한 SQL 단순화: 자주 사용하는 복잡한 SQL을 뷰로 저장 가능
 3. 논리적 독립성: 원본 테이블 변경 시에도 뷰로부터 독립적으로 접근 가능
 4. 재사용성: 다양한 쿼리에서 뷰를 호출해 재사용 가능
-

12번. 마찬가지로 뷰를 활용한 시나리오 설명

(1) 뷰 활용 시나리오 예시

```
CREATE VIEW highorders AS  
SELECT * FROM orders WHERE total_price > 500000;
```

(2) 고객 이름과 주문 도서 제목 출력 (뷰 활용)

```
SELECT c.name, o.bookname  
FROM highorders o  
JOIN customer c ON o.custid = c.custid;
```