[redacted]

*S.O.S Network - Second Project : SQLD 기출문제집 작성*

# 34회 기출문제 복원
# SQLD 예상문제


**Team** : ***S.O.S Network 월야루***

Creation Date : 2019.11.25
Last Updated : 2019.11.28
Version : 1.0

*Special Thanks to C.Y.J*

# About S.O.S Network

## 뭐하는 모임인가요?

데이터에 대해 공부하고, 탐구하여 열린 마음으로 모든 것을 공유하는 집단을 꿈꾸는 모임

SQL 과 NOSQL 모두를 아우르는 열린 마음의 데이터를 다루는 사람들의 모임 이길 ..

주요 거주지 카페 : http://cafe.naver.com/sqlpd & email 연락처 : ibutu@naver.com

## 누구누구 있나요?

현재 Oracle , AWS, Coupang, CNS 등 DB 및 Cloud 영역에서 Data 를 다루는 사람들.

앞으로는 더 많은 분야의 사람들이 있을 예정(?)

## 앞으로의 활동은?

PostgreSQL 문서에 대한 번역 Project 2탄 및 DBA 를 위한 운영 DB 관리 매뉴얼 작성 준비

SQL Coding 교육 커리큘럼 강의 준비 및 기타 교육 커리큘럼 개발 준비

## 앞의 그림은 로고 인가요?

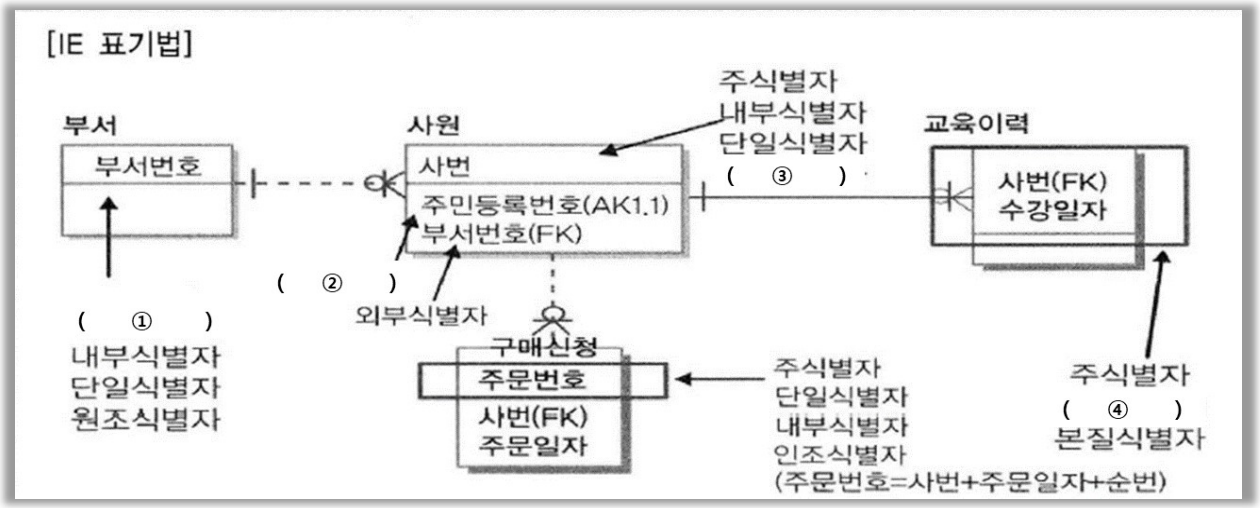
S.O.S Network 의 로고. 이 로고가 표시된 문서는 S.O.S Network의 공식 문서

# 34회 기출 문제 (2019.08.24 시험 )

**1. 도메인의 특징으로 알맞지 않은 것은?**

1) 엔터티 내에서 속성에 대한 데이터 타입과 크기를 지정한다
2) 엔터티 내에서 속성에 대한 NOT NULL 을 지정한다
3) 엔터티 내에서 속성에 대한 Check 조건을 지정한다
4) 테이블의 속성 간 FK 제약 조건을 지정한다.

**2. 아래의 그림에 대한 식별자의 분류를 알맞게 짝지은 것은?**




1) 주식별자 - 본질식별자 - 보조식별자 - 복합식별자
2) 본질식별자 - 주식별자 - 보조식별자 - 복합식별자
3) 주식별자 - 보조식별자 - 본질식별자 - 복합식별자
4) 주식별자 - 보조식별자 - 복합식별자 - 본질식별자

**3. 다음 중 주식별자를 도출하기 위한 기준으로 적절하지 않은 것은?**

1) 해당 업무에서 자주 이용되는 속성을 주식별자로 지정한다
2) 명칭, 내역 등과 같이 이름으로 기술되는 것들은 가능하면 주식별자로 지정하지 않는다
3) 복합으로 주식별자로 구성할 경우 너무 많은 속성이 포함되지 않도록 한다
4) 지정된 주식별자의 값은 변경될 수도 있다

**4. 다음 중 아래 시나리오에서 엔터티로 가장 적절한 것은 ?**

**<시나리오>**
**S 병원은 여러 명의 환자가 존재하고 각 환자에 대한 이름, 주소 등을 관리해야 한다**
**( 단, 업무범위와 데이 터 의 특성은 상기 시나리오에 기술되어 있는 사항만을 근거하여 판단해야 함)**

1) 병원
2) 환자
3) 이름
4) 주소

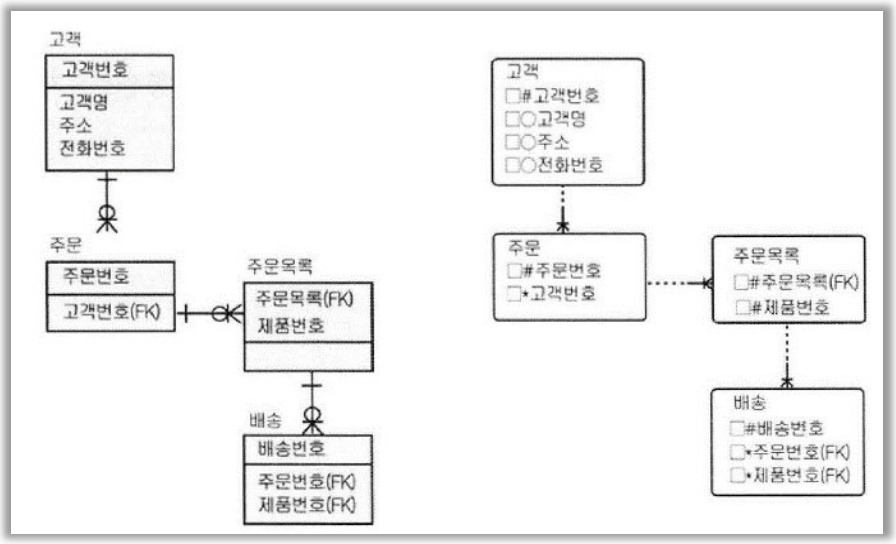
**5. 주식별자의 특징으로 가장 적절하지 않은 것은?**

1) 유일성 : 주식별자에 의해 엔터티내에서 모든 인스터스들을 유일하게 구분함
2) 최소성 : 주식별자를 구성하는 속성의 수는 유일성을 만족하는 최소의 수가 되어야 함
3) 불변성 : 주식별자가 한 번 특정 엔터티에 지정되면 그 식별자의 값은 변하지 않아야 함
4) 존재성 : 주식별자가 지정되면 데이터 값이 존재하지 않을 수 있음 (NULL 허용)

**6. 다음 중 컬럼에 대한 반정규화 기법으로 가장 적절하지 않은 것은?**

1) 중복칼럼을 추가 - 조인감소를 위해 여러 테이 블에 동일한 칼럼을 갖도록한다.
2) 파생칼럼을 추가한다 - 조회 성능을우수하게 하기 위해 미리 계산된 칼럼을 갖도록한다.
3) 이력테이블에 기능 칼럼을 추가한다 - 최신값을 처리하는 이력의 특성을 고려하여 기능성 칼럼을 추가한다
4) FK에 대한 속성을 추가한다 - FK관계 에 해당하는 속성을 추가하여 조인 성능을 높인다

**7. 아래의 ERD 에 대한 반정규화 기법으로 적절하지 않은 것은?**




1) 배송 테이블에서 고객의 정보를 찾는 빈도가 높을 경우 고객과 배송 테이블의 관계를 추가하는 관계의 반정규화를 한다.
2) 주문목록 테이블에서 고객의 정보를 찾는 빈도가 높을 경우 고객과 주문 테이블의 비식별자 관계를 식별자 관계로 한다.
3) 주문 테이블에서 항상 고객명을 같이 조회하는 경우 고객 테이블의 고객명을 주문 테이블에 넣는 컬럼의 반정규화를 한다.
4) 주문과 주문목록, 배송 테이블의 모든 컬럼을 고객 (최상위 테이블) 테이블에 모두 넣는 반정규화를 한다.

**8. 아래의 ERD를 참고하여 테이블에 대한 관계를 설명하는 것으로 가장 적절한 것은?**


1) 주문은 여러 개의 제품을 가질 수 있고, 제품은 하나의 주문에만 속할 수 있다
2) 제품은 여러 개의 주문에 속할 수 있고, 주문은 하나의 제품만 가질 수 있다
3) 주문 1개는 여러개의 제품을 가질 수 있으며, 제품 1개는 여러개의 주문에 속할 수 있다
4) 주문은 제품을 하나도 안 가질 수 있다

**9. 아래의 테이블에 대한 이상 현상에 대한 설명 중 가장 적절하지 않은 것은?**

**<SQLD_34_09>**

| 고객(PK) | 고객명 | 상품번호(PK) | 상품명 | 가격 |
|---|---|---|---|---|
| 001 | 유비 | 1000 | 스마트폰 | 100000 |
| 002 | 손권 | 1000 | 스마트폰 | 100000 |
| 003 | 관우 | 2000 | 노트북 | 5000 |
| 004 | 장비 | 3000 | LEN 카드 | 500000 |

1) 삽입이상 : 상품을 주문하지 않은 고객의 정보를 삽입할 수 없다
2) 갱신이상 : 스마트폰의 정보를 업데이트 할 경우 유비의 스마트폰만 업데이트 하면 된다
3) 갱신이상 : 노트북의 가격을 업데이트 할 경우 관우의 노트북만 업데이트 하면 된다
4) 삭제이상 : 장비의 고객정보가 삭제되면 LEN 카드 상품의 정보도 삭제된다

**10. 속성의 특징으로 가장 올바른 것은?**

1) 엔터티는 한 개의 속성만으로 구성될 수 있다
2) 엔터티를 설명하고 인스턴스의 구성요소가 된다
3) 하나의 속성에는 여러개의 속성값을 가질 수 있다
4) 속성의 특성에 따른 분류에는 PK 속성, FK 속성, 일반 속성이 있다

**11. TRUNCATE TABLE 명령어의 특징으로 가장 적절한 것은?**

1) 테이블 자체를 삭제하는 명령어로 DROP TABLE 과 동일한 명령어이다
2) 특정 로우를 선택하여 지울 수 없다
3) DELETE TABLE 과는 다르게 TRUNCATE TABLE 의 경우 정상적인 복구가 가능하다
4) DELETE TABLE 보다 시스템 부하가 더 크다

**12. 다음의 SCRIPT 를 수행한 후 보기의 SQL 을 수행할 때 잘못된 것은?**

```
<SCRIPT>
CREATE TABLE SQLD_34_12 (N1 NUMBER, N2 NUMBER );
INSERT INTO SQLD_34_12 VALUES (1,10);
INSERT INTO SQLD_34_12 VALUES (2,20);
```

1) SELECT N1 FROM SQLD_34_12 ORDER BY N2;
2) SELECT * FROM SQLD_34_12 ORDER BY 2;
3) SELECT N1 FROM (SELECT * FROM SQLD_34_12) ORDER BY N2;
4) SELECT N1 FROM (SELECT * FROM SQLD_34_12) ORDER BY 2;

**13. PROCEDURE, TRIGGER 에 대한 설명 중 가장 잘못된 것은?**

1) PROCEDURE, TRIGGER 모두 EXECUTE 명령어로 수행된다
2) PROCEDURE, TRIGGER 모두 CREATE 명령어로 생성한다
3) PROCEDURE 는 COMMIT, ROLLBACK 명령어를 사용할 수 있다
4) TRIGGER 는 COMMIT, ROLLBACK 명령어를 사용할 수 없다

**14. 아래의 데이터를 바탕으로 다음의 SQL 을 수행하였을때의 설명으로 적절하지 않은 것은?**

```
<SQL>
SELECT CONNECT_BY_ROOT LAST_NAME AS BOSS,
       MANAGER_ID,
       EMPLOYEE_ID,
       LAST_NAME,
       LEVEL,
       CONNECT_BY_ISLEAF,
       SYS_CONNECT_BY_PATH(LAST_NAME,'-') "PATH"
FROM HR.EMPLOYEES
WHERE 1=1
START WITH MANAGER_ID IS NULL
CONNECT BY PRIOR EMPLOYEE_ID = MANAGER_ID
```

**<RESULT>**

| BOSS | MANAGER_ID | EMPLOYEE_ID | LAST_NAME | "LEVEL" | CONNECT_BY_ISLEAF | PATH |
|---|---|---|---|---|---|---|
| [ ] | <null> | 100 | King | 1 | 0 | -King |
| King | 100 | 101 | Kochhar | 2 | 0 | -King-Kochhar |
| King | 101 | 108 | Greenberg | 3 | 0 | -King-Kochhar-Greenberg |
| King | 108 | 109 | Faviet | 4 | 1 | -King-Kochhar-Greenberg-Faviet |
| King | 108 | 110 | Chen | 4 | 1 | -King-Kochhar-Greenberg-Chen |
| King | 108 | 111 | Sciarra | 4 | 1 | -King-Kochhar-Greenberg-Sciarra |
| King | 108 | 112 | Urman | 4 | 1 | -King-Kochhar-Greenberg-Urman |
| King | 108 | 113 | Popp | 4 | 1 | -King-Kochhar-Greenberg-Popp |
| King | 101 | 200 | Whalen | 3 | 1 | -King-Kochhar-Whalen |
| King | 101 | 203 | Mavris | 3 | 1 | -King-Kochhar-Mavris |
| King | 101 | 204 | Baer | 3 | 1 | -King-Kochhar-Baer |
| King | 101 | 205 | Higgins | 3 | 0 | -King-Kochhar-Higgins |
| King | 205 | 206 | Gietz | 4 | 1 | -King-Kochhar-Higgins-Gietz |

1) **[ ]** 는 KING 이다
2) CONNECT_BY_ISLEAF 는 LEAF 면 1을 아니면 0 을 반환한다
3) 자식에서 부모로 가는 역방향이다
4) LEVEL 은 계층의 깊이를 의미하며 KING 은 최상위 계층이다

**15. PLAYER 테이블에서 선수명과 팀명은 오름차순, 연봉은 내림차순으로 조회하는 SQL 로 바른것은?**

1) SELECT 선수명, 팀명, 연봉 FROM ORDER BY 선수명 DESC, 팀명 DESC, 연봉 ASC
2) SELECT 선수명, 팀명, 연봉 FROM ORDER BY 선수명 ASC, 팀명 ASC , 연봉
3) SELECT 선수명, 팀명, 연봉 FROM ORDER BY 선수명 ASC, 팀명, 3 DESC
4) SELECT 선수명, 팀명, 연봉 FROM ORDER BY 선수명, 팀명, DESC 연봉

**16. 아래의 SQL에 대한 실행계획에 대한 설명으로 부적절한 것은?**

**[SQL]**
```
SELECT *
FROM HR.DEPARTMENTS A, HR.EMPLOYEES B
WHERE A.DEPARTMENT_ID = B.DEPARTMENT_ID
```

**<Plan>**
```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
| Id  | Operation          | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |             |   106 |  9540 |     6   (0)| 00:00:01 |
|*  1 |  HASH JOIN         |             |   106 |  9540 |     6   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| DEPARTMENTS |    27 |   567 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| EMPLOYEES   |   107 |  7383 |     3   (0)| 00:00:01 |
```

1) 성능향상을 위해 HASH JOIN 을 NESTED LOOP JOIN 으로 변경한다
2) SQL 의 실행 순서는 2->3->1-> 0 이다
3) DEPARTMENTS 테이블이 EMPLOYEES 보다 소량으로 선행으로 하는 것이 좋다
4) 조인 조건이 Non-Equal 일 경우 HASH JOIN 으로 수행되지 못하고 MERGE JOIN 으로 수행된다

**17. 테이블이 다음과 같을 때 아래의 SQL 연산결과로 알맞은 것은?**

**<SQL>**
**1) SELECT SUM(COL1+COL2+COL3+COL4) FROM SQLD_34_17 ;**

**2) SELECT SUM(COL1) +SUM(COL2) + SUM(COL3) + SUM(COL4)**
**FROM SQLD_34_17 ;**

**<SQLD_34_17>**

| COL1 | COL2 | COL3 | COL4 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| NULL | 1 | NULL | NULL |
| 3 | NULL | 3 | 3 |
| NULL | 4 | NULL | 4 |

1) 4, null
2) null, 22
3) 4, 22
4) null, null

**18. 아래의 SQL에서 NULL 을 반환하는 SQL은 어떤것인가?**

1) SELECT COALESCE(NULL,'2') FROM DUAL
2) SELECT NULLIF('A','A') FROM DUAL
3) SELECT NVL(NULL,0) + 10 FROM DUAL
4) SELECT NVL(NULL,'A') FROM DUAL

**19. 아래의 테이블에 대한 SQL 중 결과가 다른 하나는 무엇인가?**

**<SQLD_34_19>**

| N1 | N2 | C1 | C2 |
|---|---|---|---|
| 1 | NULL | A | NULL |
| 2 | 1 | B | A |
| 4 | 2 | D | B |
| 5 | 4 | E | D |
| 3 | 1 | C | A |

1) SELECT C1, C2, N1,N2 FROM SQLD_34_19 WHERE N1=4 START WITH N2 IS NULL CONNECT BY PRIOR N1 = N2 ;
2) SELECT C1, C2, N1,N2 FROM SQLD_34_19 START WITH C2 ='B' CONNECT BY PRIOR N1 = N2 AND C2 <>'D'
3) SELECT C1, C2, N1,N2 FROM SQLD_34_19 START WITH C1 ='B' CONNECT BY PRIOR N1 = N2 AND PRIOR C2 ='B'
4) SELECT C1, C2, N1,N2 FROM SQLD_34_19 WHERE C1 <>'B' START WITH N1 =2 CONNECT BY PRIOR N1 = N2 AND PRIOR N1 =2;

**20. 아래의 테이블에 대해 다음의 SCRIPT 를 수행한 결과로 알맞은 것은?**

**<SQL>**

```
SELECT ID, DEPT_NM, SUM(SALARY)
FROM SQLD_34_20
GROUP BY ROLLUP (ID,DEPT_NM);
```

**<SQLD_34_20>**

| ID | DEPT_NM | SALARY |
|---|---|---|
| 1 | A | 1000 |
| 1 | A | 100 |
| 2 | B | 500 |
| 2 | B | 4000 |
| 2 | B | 10 |
| 3 | C | 150 |
| 3 | C | 10 |

1)

| ID | DEPT_NM | SUM(SALARY) |
|---|---|---|
| 1 | A | 1100 |
| 2 | B | 4510 |
| 3 | C | 160 |

2)

| ID | DEPT_NM | SUM(SALARY) |
|---|---|---|
| 1 | A | 1100 |
| 1 | NULL | 1100 |
| 2 | B | 4510 |
| 2 | NULL | 4510 |
| 3 | C | 160 |
| 3 | NULL | 160 |
| NULL | NULL | 5770 |

3)

| ID | DEPT_NM | SUM(SALARY) |
|---|---|---|
| 1 | NULL | 1100 |
| 2 | NULL | 1100 |
| 3 | NULL | 160 |
| NULL | A | 1100 |
| NULL | B | 4510 |
| NULL | C | 160 |

4)

| ID | DEPT_NM | SUM(SALARY) |
|---|---|---|
| NULL | NULL | 5770 |
| NULL | A | 110 |
| NULL | B | 4510 |
| NULL | C | 160 |
| 1 | NULL | 1100 |
| 1 | A | 1100 |
| 2 | NULL | 4510 |
| 2 | B | 4510 |
| 3 | NULL | 160 |

**21. SCRIPT 를 수행한 결과가 다음과 같을 때 수행한 SQL 의 빈칸에 넣을 알맞은 그룹함수는?**

**<SQL>**

```
SELECT ID, DEPT_NM, SUM(AMT)
FROM SQLD_34_21
GROUP BY (                )
```

**<RESULT>**

| ID | DEPT_NM | "SUM(AMT)" |
|---|---|---|
| <null> | <null> | 195 |
| <null> | 가 | 25 |
| <null> | 나 | 100 |
| <null> | 다 | 70 |
| 1 | <null> | 25 |
| 1 | 가 | 25 |
| 2 | <null> | 100 |
| 2 | 나 | 100 |
| 3 | <null> | 70 |
| 3 | 다 | 70 |

1) CUBE (ID, DEPT_NM)
2) ROLLUP (ID, DEPT_NM)
3) GROUPING SETS (ID, DEPT_NM)
4) CUBE (ID)

**22. 아래의 GROUP 함수에 대한 설명으로 가장 적절한 것은 ?**

1) CUBE는 결합 가능한 모든 값에 대하여 다차원 집계를 생성한다.
2) ROLLUP 은 계층구조가 평등한 관계이므로 인수의 순서가 바뀌어도 결과는 같다.
3) ROLLUP, CUBE, GROUPING SETS 은 특정 컬럼에 대한 정렬은 가능하나 계층간 정렬은 불가능하다.
4) ROLLUP은 CUBE에 비해 시스템에 많은 부담을 주므로 사용에 주의해야 한다

**23. 아래의 테이블에 대한 SQL 결과로 올바른 것은?**

**<SQL>**

```
SELECT COUNT(*)
FROM SQLD_34_23
HAVING COUNT(*) > 4
```

**<SQLD_34_23>**

| C1 |
|---|
| 1 |
| 2 |
| 3 |
| 4 |

1) 공집합이다 (0 Rows)
2) 0
3) 1
4) 2

**24. 아래의 트랜잭션 특성에 대한 설명을 올바르게 연결한 것은?**

**<설명>**

**(ㄱ) : 트랜잭션에서 정의된 연산들은 모두 성공적으로 실행 되던지 아니면 전혀 실행되지 않은 상태로 남아 있어야 한다.**
**(ㄴ) : 트랜잭션이 실행되는 도중에 다른 트랜잭션의 영향을 받아 잘못된 결과를 만들어서는 안된다.**
**(ㄷ) : 트랜잭션이 성공적으로 수행되면 그 트랜잭션이 갱신한 데이터베이스의 내용은 영구적으로 저장된다.**
**(ㄹ) : 트랜잭션이 실행 되기 전의 데이터베이스 내용이 잘못 되어 있지 않다면 트랜잭션이 실행된 이후에도 데이터베이스의 내용에 잘못이 있으면 안된다**

1) 일관성, 원자성, 지속성, 고립성
2) 원자성, 일관성, 지속성, 고립성
3) 원자성, 고립성, 지속성, 일관성
4) 고립성, 원자성, 일관성, 지속성

**25. 아래의 테이블에 대한 SELECT 결과 건수로 알맞은 것은?**

**<SQL>**

```
SELECT DISTINCT COL1
FROM SQLD_34_25_01
UNION ALL
SELECT COL1
FROM SQLD_34_25_02
```

**<SQLD_34_25_01>**

| COL1 |
|---|
| 1 |
| 1 |
| 1 |
| 2 |
| 2 |
| 3 |
| 5 |
| 6 |

**<SQLD_34_25_02>**

| COL1 |
|---|
| 1 |
| 2 |
| 2 |
| 4 |
| 5 |

1) 4
2) 6
3) 8
4) 10

**26. 아래와 같은 테이블이 있다. 스크립트를 수행한 후의 결과로 가장 올바른 것은?**

**<SQLD_34_26_01>**

| COL1 |
|---|
| 1 |
| 2 |
| 3 |
| 4 |

**<SQLD_34_26_02>**

| COL1 |
|---|
| 1 |
| 2 |
| 3 |
| NULL |

**<SQLD_34_26_03>**

| COL1 |
|---|
| 1 |
| NULL |
| 3 |
| 5 |

**<SQLD_34_26_04>**

| COL1 |
|---|
| 1 |
| 2 |
| 5 |
| 6 |

**<SQL>**

```
SELECT COUNT(*)
FROM SQLD_34_26_01 T1 ,SQLD_34_26_02 T2 ,SQLD_34_26_03 T3 ,SQLD_34_26_04 T4
WHERE T1.COL1 = T2.COL1(+)
AND T2.COL1 = T3.COL1(+)
AND T3.COL1 = T4.COL1
```

1) 1
2) 2
3) 3
4) 4

**27. 아래 테이블에 대해 수행된 SQL 결과와 보기의 SQL 의 결과가 같은 것으로 올바른 것은?**

**<SQLD_34_27>**

| EMP_ID | DEPT_ID | SALARY |
|---|---|---|
| 1 | 10 | 1000 |
| 2 | 10 | 1500 |
| 3 | 10 | 1500 |
| 4 | 20 | 1200 |
| 5 | 20 | 1100 |
| 6 | 20 | 100 |
| 7 | 30 | 4000 |
| 8 | 30 | 5000 |

**<SQL>**

```
SELECT DEPT_ID, SALARY
FROM (
        SELECT ROW_NUMBER() OVER(PARTITION BY DEPT_ID ORDER BY SALARY DESC) RN , DEPT_ID, SALARY
        FROM SQLD_34_27 )
WHERE RN = 1
```

1)

```
SELECT DEPT_ID, SALARY
FROM (
        SELECT RANK() OVER(PARTITION BY DEPT_ID ORDER BY SALARY DESC) RN , DEPT_ID, SALARY
        FROM SQLD_34_27 )
WHERE RN = 1
```

2)

```
SELECT DEPT_ID, MAX(SALARY) AS SALARY
FROM SQLD_34_27
GROUP BY DEPT_ID
```

3)

```
SELECT DEPT_ID, SALARY
FROM SQLD_34_27
WHERE ROWNUM =1
ORDER BY DEPT_ID, SALARY DESC ;
```

4)

```
SELECT DEPT_ID, SALARY
FROM SQLD_34_27
WHERE SALARY = (SELECT MAX(SALARY) FROM SQLD_34_27 )
```

**28. 순번을 구하는 그룹함수가 아닌 것은?**

1) RANK
2) ROW_NUMBER
3) DENSE_RANK
4) RATIO_TO_REPORT

**29. 아래의 SCRIPT 에서 최종결과로 알맞은 것은?**

```
<SCRIPT>
CREATE TABLE SQLD_34_29 (N1 NUMBER, N2 NUMBER) ;
 ----------------------------------------------------------------------
INSERT INTO SQLD_34_29 VALUES (1,1);
INSERT INTO SQLD_34_29 VALUES (2,2);
SAVEPOINT SV1;

UPDATE SQLD_34_29 SET N1=4 WHERE N2=1;
SAVEPOINT SV1;

DELETE SQLD_34_29 WHERE N1 >= 2;
ROLLBACK TO SAVEPOINT SV1;

SELECT MAX(N1) FROM SQLD_34_29;
```

1) NULL
2) 4
3) 2
4) 답 없음

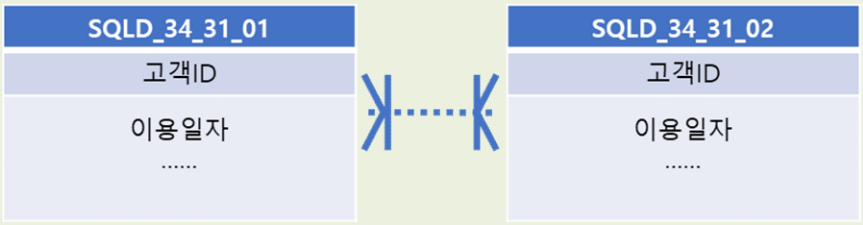
**30. 아래의 SQL 을 ORACLE 과 SQL SERVER 에서 수행할 때 SQL 에 대해 틀린 설명은? (AUTO COMMIT 은 FALSE 로 설정한다)**

```
<SCRIPT>
UPDATE SQLD_34_30 SET N1=3 WHERE N2=1;
CREATE TABLE SQLD_34_30_TEMP (N1 NUMBER);
ROLLBACK;
```

1) SQL SERVER 의 경우 ROLLBACK 이 된 후 UPDATE 와 CREATE 구문 모두 취소된다
2) SQL SERVER 의 경우 ROLLBACK 이 된 후 SQLD_34_21_TEMP 는 만들어지지 않는다.
3) ORACLE 의 경우 ROLLBACK 이 된 후 UPDATE 와 CREATE 구문 모두 취소된다.
4) ORACLE 의 경우 UPDATE 는 취소되지 않는다.

**31. 아래의 ERD 를 참고하여 보기의 SQL 중 결과가 다른 하나는?**

```
<SQL>
SELECT 고객ID, 이용일자
FROM SQLD_34_31_01
UNION
SELECT 고객ID, 이용일자
FROM SQLD_34_31_02
```




1)
```
SELECT DISTINCT
        NVL(A.고객ID ,B.고객ID) AS 고객ID,
        NVL(A.이용일자 ,B.이용일자) AS 이용일자
FROM SQLD_34_31_01 A
       CROSS JOIN SQLD SQLD_34_31_02
```

2)
```
SELECT NVL(A.고객ID,B.고객ID),
        NVL(A.이용일자,B.이용일자)
FROM SQLD_34_31_01 A
        FULL OUTER JOIN SQLD_34_31_02 B
                ON A.고객ID = B.고객ID
```

3)
```
SELECT A.고객ID,A.이용일자
FROM SQLD_34_31_01 A
         LEFT OUTER JOIN SQLD_34_31_02 B
                  ON A.고객ID = B.고객ID
UNION
SELECT B.고객ID,B.이용일자
FROM SQLD_34_31_01 A
         RIGHT OUTER JOIN SQLD_34_31_02 B
                  ON A.고객ID = B.고객ID
```

4)
```
SELECT A.고객ID,A.이용일자
FROM SQLD_34_31_01 A
WHERE NOT EXISTS (SELECT 'X'
                  FROM SQLD_34_31_02 B
                  WHERE A.고객ID = B.고객ID)
UNION
SELECT B.고객ID,B.이용일자
FROM SQLD_34_31_02 B
WHERE NOT EXISTS (SELECT 'X'
                  FROM SQLD_34_31_01 A
                  WHERE A.고객ID = B.고객ID)
```

**32. 아래와 같은 데이터를 가진 테이블이 있을 때 중복되는 이름 중 아이디가 제일 작은것만 남도록 하는 SQL 을 완성하시오**

**<SQL>**
```
DELETE SQLD_34_32
WHERE ID NOT IN (                    )
```

**<RESULT>**

| ID | NAME |
|---|---|
| 1 | A |
| 1 | C |
| 1 | D |

**<SQLD_34_32>**

| ID | NAME |
|---|---|
| 1 | A |
| 2 | A |
| 1 | C |
| 1 | D |
| 2 | D |

1) SELECT MAX(ID) FROM SQLD_34_32 GROUP BY NAME
2) SELECT MIN(ID) FROM SQLD_34_32 GROUP BY NAME
3) SELECT MAX(ID) FROM SQLD_34_32 GROUP BY ID
4) SELECT MIN(ID) FROM SQLD_34_32 GROUP BY ID

**33. 아래의 SQL 에 대한 설명 중 올바른 것은?**

**<SQL>**
```
SELECT *
FROM SQLD_34_33
WHERE EMP_NAME LIKE 'A%'
```

1) 테이블의 EMP_NAME 이 A 또는 a 로 시작하는 모든 row
2) 테이블의 EMP_NAME 이 A 로 시작하는 모든 row
3) 테이블의 EMP_NAME 이 A 로 끝나는 모든 row
4) 테이블의 EMP_NAME 이 A 또는 a 로 끝나는 모든 row

**34. 아래의 테이블 이름으로 가장 올바른 것은?**

1) TAB_100
2) 2번 SQL
3) 3번 SQL
4) 4번 SQL

**35. 반올림 함수로 알맞은 것은?**

1) ROUND
2) CEIL
3) TRUNC
4) EXP

**36. 아래의 SCRIPT 를 수행한 후 수행한 보기의 SQL 중 잘못된 것은?**

**<SCRIPT>**
```
CREATE TABLE 주문 (C1 NUMBER, C2 DATE, C3 VARCHAR2(10), C4 NUMBER DEFAULT 100 );
INSERT INTO 주문(C1,C2,C3) VALUES (1, SYSDATE, 'TEST1');
```

1) INSERT INTO 주문 VALUES (2, SYSDATE, 'TEST2')
2) DELETE 주문
3) DELETE FROM 주문
4) UPDATE 주문 SET C1=1

**37. ORDER BY 의 특징으로 가장 적절하지 않은 것은?**

1) ORDER BY 의 기본 정렬은 내림차순이다
2) SELECT 구문에 사용되지 않은 컬럼도 OERDER BY 구문에서 사용할 수 있다
3) ORDER BY 1, COL1 과 같이 숫자와 컬럼을 혼용하여 사용할 수 있다
4) ORACLE 은 NULL 을 가장 큰 값으로 취급하여 ORDER BY 시 맨 뒤로 정렬되고 SQL SERVER 반대로 가장 앞으로 정렬한다.

**38. 아래의 테이블에 대해서 다음의 SQL 을 수행한 결과로 알맞은 것은?**

**<SQLD_34_38_01>**

| 회원번호 | AMT |
|---|---|
| 1 | 60000 |
| 2 | 4000 |
| 1 | 3000 |

**<SQLD_34_38_02>**

| 등급 | MIN_AMT | MAX_AMT |
|---|---|---|
| VVIP | 10001 | 100000 |
| VIP | 1000 | 10000 |
| SILVER | 100 | 999 |

```
<SQL>
SELECT A.회원번호, B.등급
FROM (SELECT 회원번호, SUM(AMT)
        FROM SQLD_34_38_01
        GROUP BY 회원번호) A , SQLD_34_38_02 B
WHERE 1=1
AND A.AMT BETWEEN B.MIN_AMT AND B.MAX_AMT
```

1)

| 회원번호 | 등급 |
|---|---|
| 1 | VVIP |
| 2 | VIP |

2)

| 회원번호 | 등급 |
|---|---|
| 1 | VIP |
| 2 | SILVER |

3)

| 회원번호 | 등급 |
|---|---|
| 1 | VVIP |
| 2 | SILVER |

4)

| 회원번호 | 등급 |
|---|---|
| 1 | VIP |
| 2 | VVIP |

**39. 조인 기법 설명중 가장 적절한 것은?**

1) Hash Join 은 정렬 작업이 없어 정렬이 부담되는 대량배치작업에 유리하다.
2) 대용량의 데이터를 가진 두개 테이블을 조인할 때 Hash Join 보다 Nested Loop Join 이 더 유리하다
3) 옵티마이저는 조인컬럼에 인덱스가 존재하지 않으면 Nested Loop Join 을 선호한다.
4) Nested Loop Join 기법은 배치작업에서 선호하는 조인기법이다.

**40. 사원과 관리자, 그리고 최상위 관리자 나오도록 작성된 SQL 을 완성하시오.**

```
<SQL >
SELECT A.EMPLOYEE_ID,
       A.MANAGER_ID AS A_MANAGER_ID,
       B.EMPLOYEE_ID AS B_EMPLOYEE_ID,
       B.MANAGER_ID AS B_MANAGER_ID,
       A.LAST_NAME
FROM HR.EMPLOYEES A
      (          ) HR.EMPLOYEES B ON (          )
WHERE 1=1
AND A.EMPLOYEE_ID < 200
ORDER BY EMPLOYEE_ID ;
```

1) INNER JOIN , A.MANAGER_ID = B.EMPLOYEE_ID
2) INNER JOIN , A.EMPLOYEE_ID = B.MAANGER_ID
3) LEFT OUTER JOIN , A.MANAGER_ID = B.EMPLOYEE_ID
4) LEFT OUTER JOIN , A.EMPLOYEE_ID = B.MAANGER_ID

**41. SQL 집합 연산자 INTERSECT 에 대한 설명 중 올바른 것은?**

1) 결과의 합집합으로 중복된 행을 모두 포함한다.
2) 결과의 합집합으로 중복된 행은 하나의 행으로 표시한다
3) 결과의 교집합으로 중복된 행을 하나의 행으로 표시한다
4) 결과의 교집합으로 중복된 행을 모두 포함한다

**42. 아래의 Window function 에 대한 설명중 적절한 것은?**

1) Partition 과 Group By 구문은 의미적으로 완전히 다르다
2) Sum,max, min 등과 같은 집계 window function을 사용할 때 window 절과 함께 사용하면 집계의 대상이 되는 레코드 범위를 지정할 수 있다
3) Window function 처리로 인해 결과 건수가 줄어들 수 있다
4) GROUP BY 구문과 Window function 은 병행하여 사용 할 수 있다

43. 부서에 대한 정보를 보여주고자 한다. 부서명, 부서에 소속된 사원명, 부서번호를 보여주고자 하며,
사원이 없는 부서도 보여주고자 할 때 아래의 SQL 을 완성하시오. (  )

```
<SQL>
SELECT A.DEPT_NM, B.EMP_NM, A.DEPT_ID
FROM DEPT A (  ) EMP B ON (A.DEPT_ID = B.DEPT_ID )
```

44. 아래와 같은 테이블이 있을 때 아래의 SQL 을 수행했을 때 두번째로 나오는 값은 무엇인가? (  )

```
<SQL>
SELECT CODE
FROM SQLD_34_44
START WITH SUPER_ID IS NULL
CONNECT BY PRIOR ID = SUPER_ID
ORDER SIBLINGS BY CODE DESC;
```

<SQLD_34_44>

| ID | SUPER_ID | CODE |
|---|---|---|
| 1 | NULL | A |
| 2 | 1 | B |
| 3 | 1 | C |
| 4 | 2 | D |

45. 아래와 같은 테이블이 있을 때 연봉이 2번째, 3번째로 높은 사원의 정보를 구하고자 한다. 아래의 SQL 을 완성하시오.

<SQLD_34_45>

| EMPLOYEE_ID | LAST_NAME | SALARY |
|---|---|---|
| 100 | King | 24000 |
| 101 | Kochhar | 17000 |
| 103 | Hunold | 9000 |
| 104 | Ernst | 6000 |
| 105 | Austin | 4800 |

```
<SQL>
SELECT *
FROM (
   SELECT EMPLOYEE_ID, LAST_NAME, SALARY, RANK() OVER(ORDER BY SALARY DESC ) RN
   FROM SQLD_34_45
   WHERE 1=1
   AND SALARY < (SELECT (  ) (SALARY) FROM EMPLOYEES )  )
WHERE RN < 3;
```

46. SELECT UPPER(sqldeveloper) FROM DUAL 의 결과를 적으시오. (  )

47. 아래와 같은 데이터를 가진 테이블이 있을 때 SQL 의 수행 결과를 적으시오. (  )

<SQLD_34_47>

| EMPLOYEE_ID | LAST_NAME | MANAGER_ID | SALARY |
|---|---|---|---|
| 100 | King | | 24000 |
| 101 | Kochhar | 100 | 17000 |
| 102 | De Haan | 100 | 17000 |
| 103 | Hunold | 102 | 9000 |
| 104 | Ernst | 103 | 6000 |
| 105 | Austin | 103 | 4800 |
| 106 | Pataballa | 103 | 4800 |
| 107 | Lorentz | 103 | 4200 |
| 108 | Greenberg | 101 | 12000 |
| 109 | Faviet | 108 | 9000 |

```
<SQL>
WITH WITH_TAB (last_name, EMP_ID, MGR_ID, sum_salary  )
AS (
 SELECT last_name,EMPLOYEE_ID,MANAGER_ID, salary
 FROM HR.EMPLOYEES
 WHERE MANAGER_ID IS NULL
 UNION ALL
 SELECT a.last_name, a.EMPLoYEE_ID, a.MANAGER_ID, a.salary + b.sum_salary
 FROM  HR.EMPLOYEES A, WITH_TAB B
 WHERE B.EMP_ID = A.MANAGER_ID   )
SELECT SUM_SALARY FROM WITH_TAB
WHERE EMP_ID = 105;
```

**48. 아래와 같은 데이터를 가진 테이블이 있을 때 SQL 의 결과를 작성하시오. (  )**

```
<SQL>
SELECT COUNT(*)
FROM SQLD_34_48
WHERE SALARY > 200
    OR MGR_ID IS NULL
  AND CODE ='B'
```

<SQLD_34_48>

| EMP_ID | MGR_ID | CODE | SALARY |
|---|---|---|---|
| 1 | 2 | A | 100 |
| 2 | 5 | B | 300 |
| 3 | NULL | CODE | 150 |
| 4 | 1 | D | 400 |
| 5 | 7 | E | 500 |

**49. 아래와 같은 데이터를 가진 테이블에 대한 SQL 을 수행했을 때 결과가 다음과 같다. SQL 을 완성하시오. (  )**

```
<SQL>
SELECT VAL, COUNT(*) AS CNT
FROM (
     SELECT (          )(4) OVER (ORDER BY COL1) AS VAL
     FROM SQLD_34_X7
    )
WHERE 1=1
GROUP BY VAL
ORDER BY 1
```

<SQLD_34_49>

| COL1 |
|---|
| A |
| B |
| C |
| D |
| E |
| F |
| G |
| H |
| I |
| J |

<RESULT>

| VAL | CNT |
|---|---|
| 1 | 3 |
| 2 | 3 |
| 3 | 2 |
| 4 | 2 |

**50. 아래의 sql 결과를 출력하는 SQL 을 완성하시오. (  )**

<RESULT>

| EMPLOYEE_ID | DEPARTMENT_ID | LAST_NAME | SALARY | BEFORE_SALARY |
|---|---|---|---|---|
| 107 | 60 | Lorentz | 4200.00 | <null> |
| 106 | 60 | Pataballa | 4800.00 | <null> |
| 105 | 60 | Austin | 4800.00 | 4200 |
| 104 | 60 | Ernst | 6000.00 | 4800 |
| 103 | 60 | Hunold | 9000.00 | 4800 |
| 102 | 90 | De Haan | 17000.00 | <null> |
| 101 | 90 | Kochhar | 17000.00 | <null> |
| 100 | 90 | King | 24000.00 | 17000 |
| 109 | 100 | Faviet | 9000.00 | <null> |
| 108 | 100 | Greenberg | 12000.00 | <null> |

```
<SQL>
SELECT EMPLOYEE_ID,
       DEPARTMENT_ID,
       LAST_NAME,
       SALARY,
       LAG(SALARY, (     )    ) OVER(PARTITION BY DEPARTMENT_ID ORDER BY SALARY ) AS BEFORE_SALARY
FROM HR.EMPLOYEES
WHERE EMPLOYEE_ID < 110;
```