

해당 창작물의 허가되지 않은 재배포 , 무단 복사 및 무단 게시를 엄격히 금지합니다.
반드시 저작자에게 문의한 후 허가를 받아 사용해주세요.

(자료 중 빨간글씨로 작성된 숫자는 [SQL 자격검정 실전문제 , 한국데이터산업진흥원] 책의 문항을 의미하며 해당 개념으로 관련된 문항이라고 해석을 하시면 됩니다.)

SQLD 특강

21 ~ 28 SQL 활용 PART2

강사 강태우

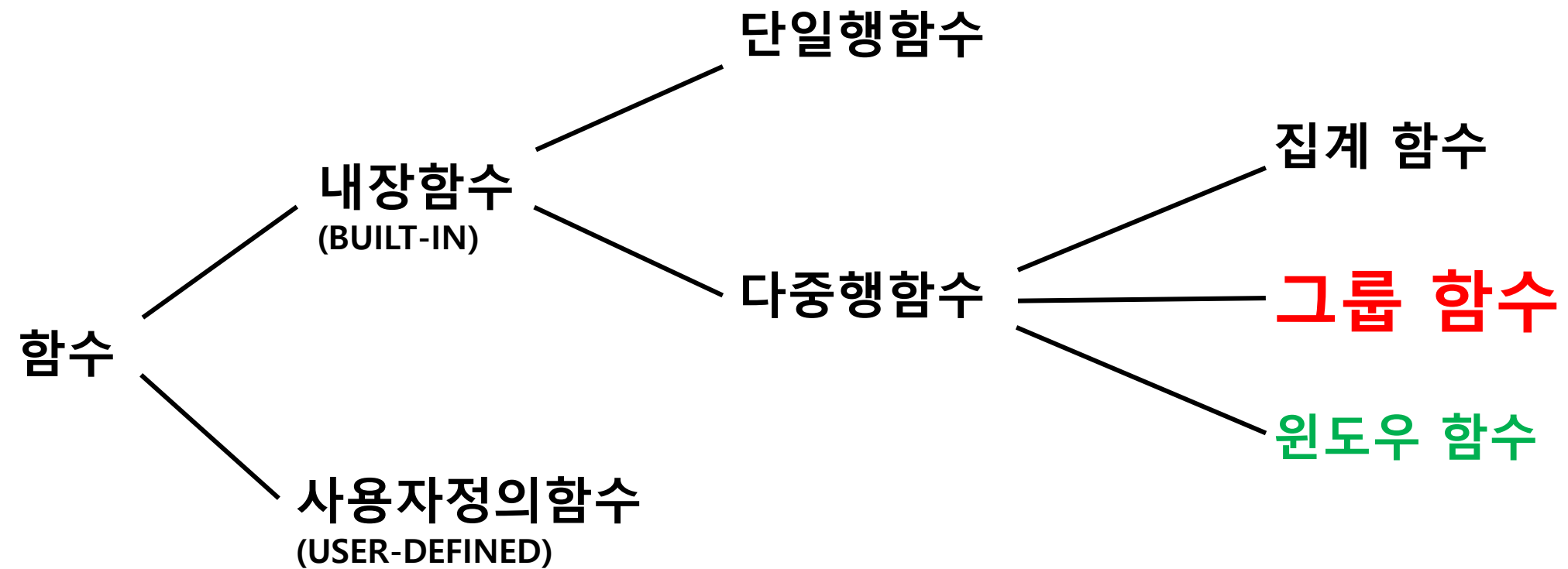
SQLD 특강

- 25. [그룹 함수](#)
- 26. [윈도우 함수](#)
- 27. [DCL](#)
- 28. [절차형 SQL](#)

25. 그룹 함수

105 ~ 111

함수의 종류



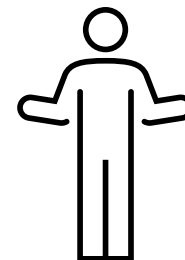
그룹 함수 종류 3가지

ROLLUP

CUBE

GROUPING SETS

이 3가지 함수를 이용해서
다차원의 관점에서 집계처리
를 할 수 있습니다.



직원 테이블에서 부서별 연봉합과 전체 연봉합을 추출해봅시다 (GROUP BY 활용)

직원ID	이름	부서ID	연봉
A0001	김철수	D001	4800
A0002	강홍수	D002	(null)
A0003	이현정	D003	2600
A0004	김선미	D004	4500
A0005	문헌철	D005	5000
A0006	송대주	D001	7500
A0007	메이슨	D002	6200
A0008	송진아	D003	7500
A0009	이서연	D004	9000
A0010	김홍민	D005	9300

부서별 연봉합



```
SELECT 부서ID , SUM(연봉)
FROM 직원
GROUP BY 부서ID ;
```

부서ID	SUM(연봉)
D001	12300
D002	6200
D003	10100
D004	13500
D005	14300

전체 연봉합




```
SELECT SUM(연봉)
FROM 직원 ;
```

SUM(연봉)
56400

ROLLUP을 이용하면 소계와 합계를 한번에 추출할 수 있습니다.

즉 **한번의 작성으로 여러 집계**를 낼 수 있게 됩니다.

직원ID	이름	부서ID	연봉
A0001	김철수	D001	4800
A0002	강홍수	D002	(null)
A0003	이현정	D003	2600
A0004	김선미	D004	4500
A0005	문현철	D005	5000
A0006	송대주	D001	7500
A0007	메이슨	D002	6200
A0008	송진아	D003	7500
A0009	이서연	D004	9000
A0010	김홍민	D005	9300

부서별 연봉합
+
전체 연봉합


```
SELECT 부서ID , SUM(연봉)
FROM 직원
GROUP BY ROLLUP (부서ID) ;
```

부서ID	SUM(연봉)
D001	12300
D002	6200
D003	10100
D004	13500
D005	14300
(null)	56400

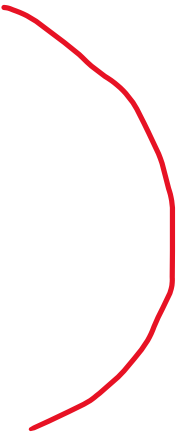


ROLLUP 을 GROUP BY 로 변경하면 다음과 같습니다.

```
GROUP BY ROLLUP (A , B) ;
```

||

```
GROUP BY A, B ;  
GROUP BY A ;  
GROUP BY (전체)
```



앞에서부터 계층적으로 하나씩 사라집니다.

예시 데이터로 확인해봅시다. (ROLLUP)

< 고객월별가스사용량 >

고객ID	사용월	사용량
A0001	01	100
A0001	02	120
A0001	03	150
A0002	01	120
A0002	02	150
A0002	03	200
A0003	03	400

```
SELECT 고객ID , 사용월 , SUM(사용량)
FROM 고객월별가스사용량
GROUP BY ROLLUP ( 고객ID , 사용월 ) ;
```

고객ID	사용월	SUM(사용량)
A0001	01	100
A0001	02	120
A0001	03	150
A0002	01	120
A0002	02	150
A0002	03	200
A0003	03	400
A0001	(null)	370
A0002	(null)	470
A0003	(null)	400
(null)	(null)	1240

ROLLUP 을 GROUP BY 로 표현해봅시다.

< 고객월별가스사용량 >

고객ID	사용월	사용량
A0001	01	100
A0001	02	120
A0001	03	150
A0002	01	120
A0002	02	150
A0002	03	200
A0003	03	400

```
SELECT 고객ID , 사용월 , SUM(사용량)
  FROM 고객월별가스사용량
 GROUP BY ROLLUP ( 고객ID , 사용월 ) ;
```

=

```
-- 고객별 월별 가스 사용량 합계
SELECT 고객ID , 사용월 , SUM(사용량)
  FROM 고객월별가스사용량
 GROUP BY 고객ID , 사용월

UNION ALL

-- 고객별 가스 사용량 합계
SELECT 고객ID , NULL , SUM(사용량)
  FROM 고객월별가스사용량
 GROUP BY 고객ID

UNION ALL

-- 고객 전체 사용량 합계
SELECT NULL , NULL, SUM(사용량)
  FROM 고객월별가스사용량;
```

CUBE는 입력된 컬럼의 모든 경우의 수를 집계합니다.

< 고객월별가스사용량 >

고객ID	사용월	사용량
A0001	01	100
A0001	02	120
A0001	03	150
A0002	01	120
A0002	02	150
A0002	03	200
A0003	03	400



고객ID	사용월	SUM(사용량)
(null)	(null)	1240
(null)	01	220
(null)	02	270
(null)	03	750
A0001	(null)	370
A0001	01	100
A0001	02	120
A0001	03	150
A0002	(null)	470
A0002	01	120
A0002	02	150
A0002	03	200
A0003	(null)	400
A0003	03	400

```
SELECT 고객ID , 사용월 , SUM(사용량)
FROM 고객월별가스사용량
GROUP BY CUBE ( 고객ID , 사용월 ) ;
```

CUBE를 GROUP BY 로 변경하면 다음과 같습니다.

```
GROUP BY CUBE ( A, B ) ;
```

||

```
GROUP BY A, B ;  
GROUP BY A ;  
GROUP BY B ;  
GROUP BY (전체) ;
```

입력된 컬럼의 2^N 개 만큼 group by 가 발생합니다.

CUBE를 GROUP BY 로 표현해봅시다.

< 고객월별가스사용량 >

고객ID	사용월	사용량
A0001	01	100
A0001	02	120
A0001	03	150
A0002	01	120
A0002	02	150
A0002	03	200
A0003	03	400

```
SELECT 고객ID , 사용월 , SUM(사용량)
FROM 고객월별가스사용량
GROUP BY CUBE ( 고객ID , 사용월 ) ;
```

=

```
-- 고객별 월별 가스 사용량 합계
SELECT 고객ID , 사용월 , SUM(사용량)
FROM 고객월별가스사용량
GROUP BY 고객ID , 사용월

UNION ALL

-- 고객별 가스 사용량 합계
SELECT 고객ID , NULL , SUM(사용량)
FROM 고객월별가스사용량
GROUP BY 고객ID

UNION ALL

-- 사용월별 가스 사용량 합계
SELECT NULL , 사용월 , SUM(사용량)
FROM 고객월별가스사용량
GROUP BY 사용월

UNION ALL

-- 고객 전체 사용량 합계
SELECT NULL , NULL , SUM(사용량)
FROM 고객월별가스사용량;
```

GROUPING SETS 은 입력한 컬럼 그대로 집계처리를 합니다.

< 고객월별가스사용량 >

고객ID	사용월	사용량
A0001	01	100
A0001	02	120
A0001	03	150
A0002	01	120
A0002	02	150
A0002	03	200
A0003	03	400



고객ID	사용월	SUM(사용량)
(null)	03	750
(null)	02	270
(null)	01	220
A0002	(null)	470
A0001	(null)	370
A0003	(null)	400

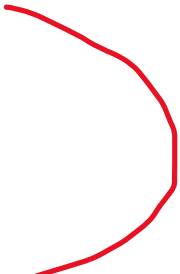
```
SELECT 고객ID , 사용월 , SUM(사용량)
FROM 고객월별가스사용량
GROUP BY GROUPING SETS(고객ID , 사용월) ;
```

GROUPING SETS 를 GROUP BY 로 변경하면 다음과 같습니다.

```
GROUP BY GROUPING SETS ( A , B ) ;
```

||

```
GROUP BY A ;  
GROUP BY B ;
```



말 그대로 입력한 컬럼을 하나씩 GROUP BY 처리 합니다.

GROUPING SETS 를 GROUP BY 로 표현해봅시다.

< 고객월별가스사용량 >

고객ID	사용월	사용량
A0001	01	100
A0001	02	120
A0001	03	150
A0002	01	120
A0002	02	150
A0002	03	200
A0003	03	400

=

```
SELECT 고객ID , NULL , SUM(사용량)
  FROM 고객월별가스사용량
 GROUP BY 고객ID

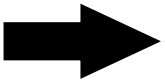
UNION ALL

SELECT NULL , 사용월 , SUM(사용량)
  FROM 고객월별가스사용량
 GROUP BY 사용월 ;
```

```
SELECT 고객ID , 사용월 , SUM(사용량)
  FROM 고객월별가스사용량
 GROUP BY GROUPING SETS (고객ID , 사용월) ;
```


NULL로 표시되는 부분을 바꿔줄 수 있습니다. [GROUPING()]

```
SELECT 고객ID , 사용월 , SUM(사용량)
FROM 고객월별가스사용량
GROUP BY ROLLUP( 고객ID , 사용월 ) ;
```



```
SELECT CASE WHEN GROUPING(고객ID) = 1 AND GROUPING(사용월) = 1
THEN '전체'
ELSE 고객ID
END AS 고객ID
, CASE WHEN GROUPING(사용월) = 1
THEN '소계'
ELSE 사용월
END AS 사용월
, SUM(사용량)
FROM 고객월별가스사용량
GROUP BY ROLLUP( 고객ID , 사용월 ) ;
```

GROUP BY 에 사용중이지 않으면 1 , 사용중이면 0

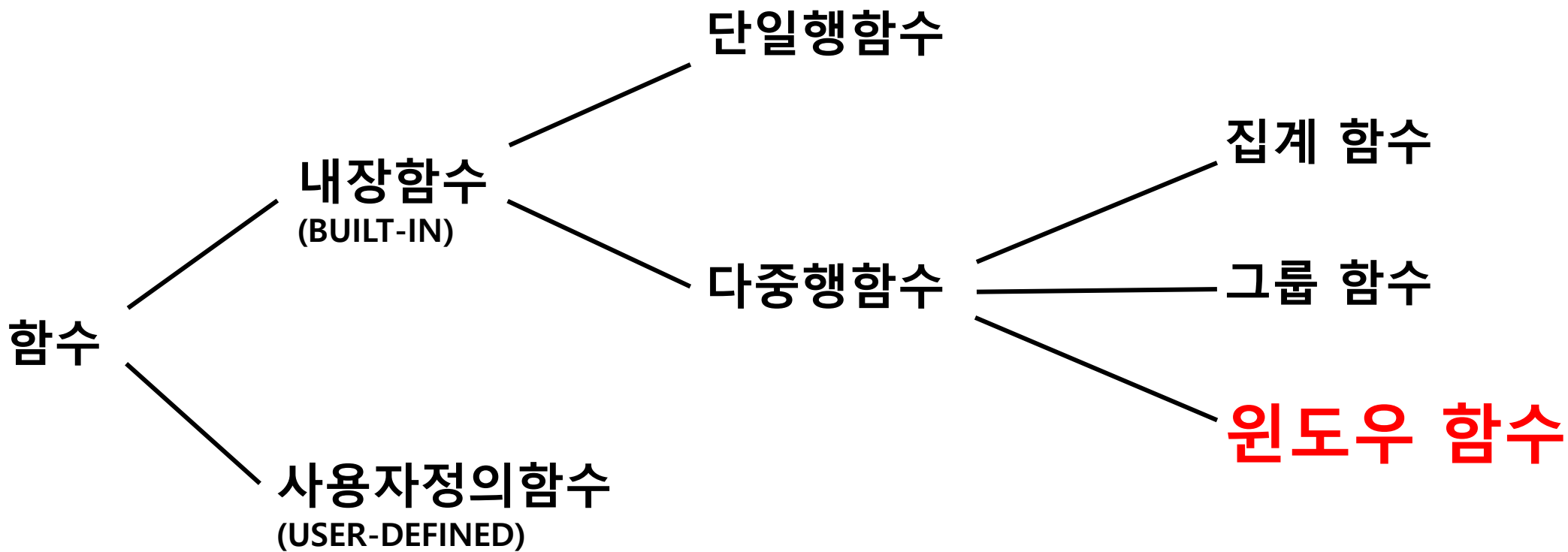
고객ID	사용월	SUM(사용량)
A0001	01	100
A0001	02	120
A0001	03	150
A0002	01	120
A0002	02	150
A0002	03	200
A0003	03	400
A0001	(null)	370
A0002	(null)	470
A0003	(null)	400
(null)	(null)	1240

고객ID	사용월	SUM(사용량)
A0001	01	100
A0001	02	120
A0001	03	150
A0002	01	120
A0002	02	150
A0002	03	200
A0003	03	400
A0001	소계	370
A0002	소계	470
A0003	소계	400
전체	소계	1240

26. 윈도우 함수

112 ~ 118

함수의 종류



윈도우 함수란?

테이블의 행과 행간의 관계를 정의, 비교, 연산할 때 쓸 수 있는 함수 종류

순위 함수 : RANK , DENSE_RANK , ROW_NUMBER

그룹 내 집계 함수 : SUM , MAX , MIN , AVG , COUNT

그룹 내 행 순서 함수 : FIRST_VALUE , LAST_VALUE , LAG , LEAD

비율 관련 함수 : CUME_DIST , PERCENT_RANK , NTILE



윈도우 함수 문법

WINDOW_FUNCTION (매개변수) OVER ([PARTITION BY 컬럼] [ORDER BY 절] [WINDOWING 절])

WINDOW_FUNCTION : 사용할 윈도우 함수를 작성합니다. 매개변수가 필요하면 작성합니다.

OVER : 윈도우함수에 반드시 사용되는 구절입니다.

PARTITION BY 컬럼 : 테이블 내부의 행들을 특정 컬럼을 기준으로 그룹화합니다 (GROUP BY 와 비슷)

ORDER BY 컬럼 : 특정 컬럼 기준으로 그룹화(PARTITION BY) 한 대상을 정렬합니다.

WINDOWING 절 : 테이블 내에서 사용하려는 행의 범위를 지정합니다.

순위 함수 (RANK , DENSE_RANK , ROW_NUMBER)

테이블의 행과 행간의 관계를 **순위로 비교**할 수 있다.

WINDOW_FUNCTION (매개변수) OVER ([PARTITION BY 컬럼] [ORDER BY 절] [WINDOWING 절])

<지점>

지점코드	지점위치
001	용봉점
002	매곡점
003	충잠점
004	송정점
005	화정점

<월별매출>

년월	지점코드	매출액
202201	001	200
202201	002	300
202201	003	250
202201	004	300
202201	005	250
202202	001	150
202202	002	150
202202	004	200
202202	005	100
202203	002	100
202203	003	100
202203	004	200
202203	005	350

Q.실행 결과는 무엇인가?

```
SELECT 지점코드
      , ( SELECT 지점위치
            FROM 지점
            WHERE 지점코드 = x.지점코드 ) AS 지점위치
      , 매출액
      , RANK() OVER ( ORDER BY 매출액 DESC ) AS 순위
FROM (
      SELECT A.지점코드
            , SUM(B.매출액) AS 매출액
      FROM 지점 A INNER JOIN 월별매출 B
      ON ( A.지점코드 = B.지점코드 )
      GROUP BY A.지점코드
    ) X
ORDER BY 순위 ;
```

▶ 26. 윈도우 함수

<지점>

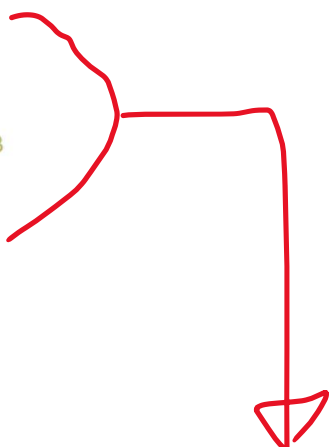
지점코드	지점위치
001	용봉점
002	매곡점
003	충잠점
004	송정점
005	화정점

<월별매출>

년월	지점코드	매출액
202201	001	200
202201	002	300
202201	003	250
202201	004	300
202201	005	250
202202	001	150
202202	002	150
202202	004	200
202202	005	100
202203	002	100
202203	003	100
202203	004	200
202203	005	350

Q.실행 결과는 무엇인가?

```
SELECT 지점코드
, ( SELECT 지점위치
    FROM 지점
    WHERE 지점코드 = x.지점코드 ) AS 지점위치
, 매출액
, RANK() OVER ( ORDER BY 매출액 DESC ) AS 순위
FROM (
    SELECT A.지점코드
    , SUM(B.매출액) AS 매출액
    FROM 지점 A INNER JOIN 월별매출 B
    ON ( A.지점코드 = B.지점코드 )
    GROUP BY A.지점코드
) X
ORDER BY 순위 ;
```



지점코드	매출액
005	700
004	700
002	550
001	350
003	350

순위 함수 (RANK , DENSE_RANK , ROW_NUMBER) 차이

RANK – 동 순위가 있으면 순위를 건너뛰고 부여한다.

지점코드	지점위치	매출액	순위
005	화정점	700	1
004	송정점	700	1
002	매곡점	550	3
001	용봉점	350	4
003	충암점	350	4

DENSE_RANK – 동 순위가 있어도 순위를 건너 뛰지 않는다.

지점코드	지점위치	매출액	순위
005	화정점	700	1
004	송정점	700	1
002	매곡점	550	2
001	용봉점	350	3
003	충암점	350	3

ROW_NUMBER – 동 순위든 아니든 고유한 번호를 부여한다.

지점코드	지점위치	매출액	순위
005	화정점	700	1
004	송정점	700	2
002	매곡점	550	3
001	용봉점	350	4
003	충암점	350	5

순위 함수 (RANK , DENSE_RANK , ROW_NUMBER)

테이블의 행과 행간의 관계를 순위로 비교할 수 있다.

WINDOW_FUNCTION (매개변수) OVER ([PARTITION BY 컬럼] [ORDER BY 절] [WINDOWING 절])

<월드컵경기내역>

경기장	승자	패자	점수차
카타르	A팀	B팀	5
카타르	A팀	C팀	3
카타르	A팀	D팀	2
독일	E팀	F팀	10
독일	E팀	G팀	1
독일	E팀	H팀	4
대한민국	I팀	J팀	6
대한민국	I팀	K팀	7

Q.실행 결과는 무엇인가?

```
SELECT 경기장 , 승자 , 패자 , 점수차
FROM (
    SELECT 경기장 , 승자 , 패자 , 점수차
        , ROW_NUMBER() OVER ( PARTITION BY 경기장 ORDER BY 점수차 DESC ) AS RNUM
    FROM 월드컵경기내역
)
WHERE RNUM = 1;
```

<월드컵경기내역>

⚽ 경기장	⚽ 승자	⚽ 패자	⚽ 점수차
카타르	A팀	B팀	5
카타르	A팀	C팀	3
카타르	A팀	D팀	2
북아일랜드	E팀	F팀	10
북아일랜드	E팀	G팀	1
북아일랜드	E팀	H팀	4
대한민국	I팀	J팀	6
대한민국	I팀	K팀	7

Q.실행 결과는 무엇인가?

```
SELECT 경기장 , 승자 , 패자 , 점수차
FROM (
    SELECT 경기장 , 승자 , 패자 , 점수차
        , ROW_NUMBER() OVER ( PARTITION BY 경기장 ORDER BY 점수차 DESC ) AS RNUM
    FROM 월드컵경기내역
)
WHERE RNUM = 1;
```

순위 함수 (RANK , DENSE_RANK , ROW_NUMBER)

테이블의 행과 행간의 관계를 순위로 비교할 수 있다.

WINDOW_FUNCTION (매개변수) OVER ([PARTITION BY 컬럼] [ORDER BY 절] [WINDOWING 절])

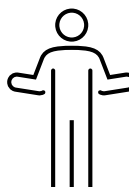
<상품>

분류코드	상품명	가격
001	상품1	5000
001	상품2	8000
001	상품3	11000
001	상품4	35000
002	상품5	25000
002	상품6	25000
002	상품7	15000
002	상품8	25000
002	상품9	20000
003	상품10	5000
003	상품11	5000
003	상품12	5000
004	상품13	40000
004	상품14	50000

Q.실행 결과는 무엇인가?

```
SELECT 분류코드
      , 상품명
      , 가격
      , SUM(가격) OVER ( ORDER BY 분류코드 , 상품명 RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS 현재행까지합계
FROM 상품
```

WINDOWING 범위 설정시
ORDER BY 는 필수입니다!!



▶ 26. 윈도우 함수

<상품>

분류코드	상품명	가격
001	상품1	5000
001	상품2	8000
001	상품3	11000
001	상품4	35000
002	상품5	25000
002	상품6	25000
002	상품7	15000
002	상품8	25000
002	상품9	20000
003	상품10	5000
003	상품11	5000
003	상품12	5000
004	상품13	40000
004	상품14	50000

Q.실행 결과는 무엇인가?

```
SELECT 분류코드
      , 상품명
      , 가격
      , SUM(가격) OVER ( ORDER BY 분류코드 , 상품명 RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW ) AS 현재행까지합계
FROM 상품
```

WINDOWING 절 범위

WINDOW_FUNCTION (매개변수) OVER ([PARTITION BY 컬럼] [ORDER BY 절] [WINDOWING 절])

지점	일자	수입	첫~끝범위	첫~현재행범위	-10000~+10000	앞의행1개~뒤의행1개
광주	2023.01.06	10000	100000	10000	45000	25000
광주	2023.01.07	15000	100000	25000	70000	45000
광주	2023.01.08	20000	100000	45000	100000	60000
광주	2023.01.09	25000	100000	70000	90000	75000
광주	2023.01.10	30000	100000	100000	75000	55000
서울	2023.01.01	10000	150000	10000	30000	30000
서울	2023.01.02	20000	150000	30000	60000	60000
서울	2023.01.03	30000	150000	60000	90000	90000
서울	2023.01.04	40000	150000	100000	120000	120000
서울	2023.01.05	50000	150000	150000	90000	90000

* 아래 슬라이드의 쿼리를 실행해보세요.

LEAD 와 LAG 함수에 대해 알아봅시다.

LEAD 와 LAG 는 각각 파티션별 윈도우에서 이전/이후 N 번째의 행의 값을 가져올 수 있습니다.

CODE	COL1	COL2
C001	1	2
C001	2	3
C001	3	5
C001	4	6
C001	5	8

LEAD(컬럼 , N , DEFAULT) : N번째 이후 행의 컬럼값을 가져오고 없다면 DEFAULT 값을 출력

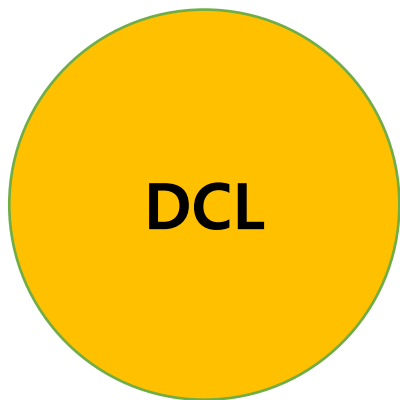
LAG(컬럼 , N , DEFAULT) : N번째 이전 행의 컬럼값을 가져오고 없다면 DEFAULT 값을 출력

```
SELECT CODE
      , COL1
      , COL2
      , LAG(COL1) OVER ( ORDER BY COL1 ) AS LAG값
      , LEAD(COL2) OVER ( ORDER BY COL2 ) AS LEAD값
FROM TAB1 ;
```

27. DCL (Data Control Language)

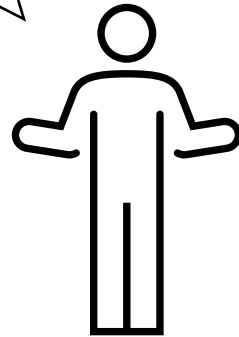
119, 120, 121, 122

DCL(데이터제어어) 란?



객체를 제어한다.

데이터베이스는 권한을
부여/회수 하면서
객체를 보호합니다.



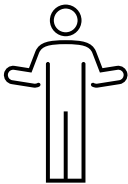
GRANT
REVOKE
ROLE

GRANT 와 REVOKE

▶ GRANT는 권한을 부여하는 명령입니다.

grant 부여할 권한 [on 대상객체] to 부여받을계정 [with grant option]

내가 USRE_B의 TAB1 테이블을
조회하고 변경할 수 있을까?



USER_A

NO! 너는 SELECT 할 권한이 없어 .

```
SELECT *  
FROM USER_B.TAB1 ;
```

질의 결과 x

SQL | 실행 중:SELECT * FROM C##USER_B,TAB1(0초)

ORA-00942: 테이블 또는 뷰가 존재하지 않습니다
00942, 00000 - "table or view does not exist"



USER_B

TAB1	
COL1	COL2
A	100
B	200
C	300

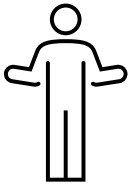
GRANT 와 REVOKE

▶ GRANT는 권한을 부여하는 명령입니다.

grant 부여할권한 [on 대상객체] to 부여받을계정 [with grant option]

```
SQL> GRANT SELECT ON USER_B.TAB1 TO USER_A ;
권한이 부여되었습니다.
```

내가 USRE_B의 TAB1 테이블을
조회하고 변경할 수 있을까?



USER_A

SELECT *	
FROM USER_B.TAB1 ;	
의 결과 x	
SQL 인출된 모든 행: 3(0.02초)	
COL1	COL2
1 A	100
2 B	200
3 C	300



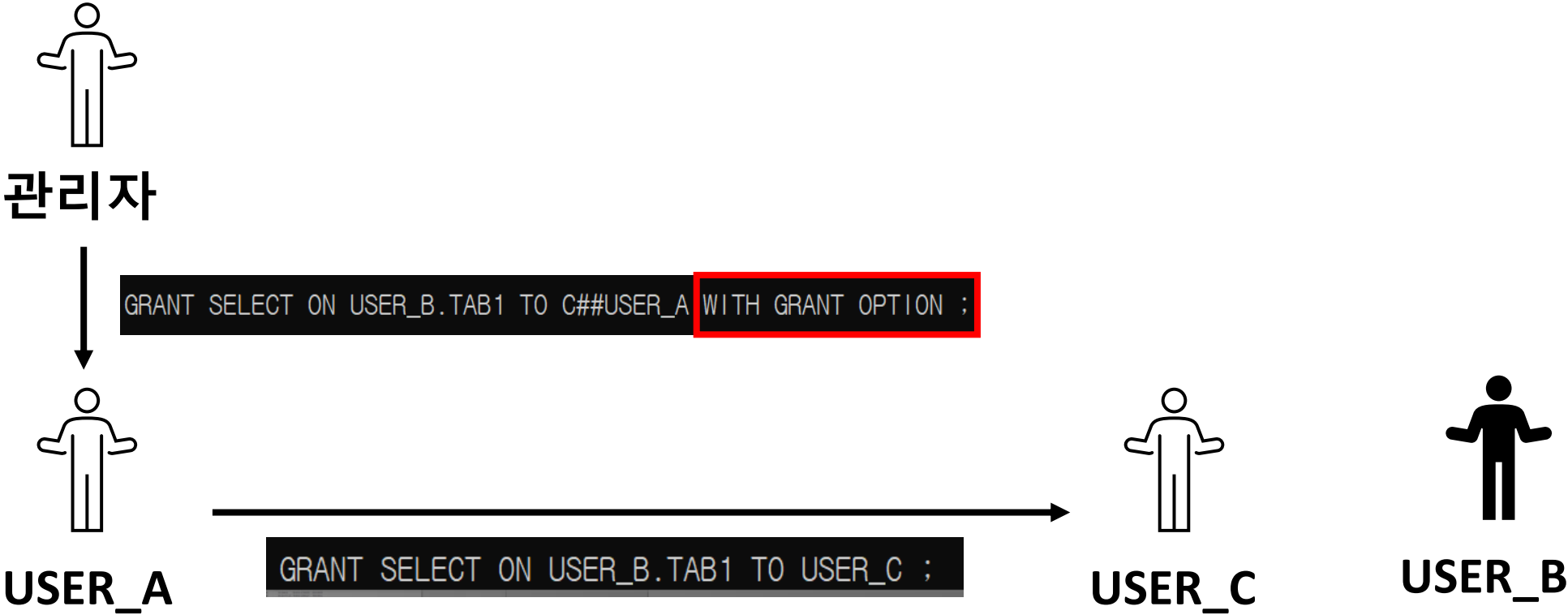
USER_B

TAB1	
COL1	COL2
A	100
B	200
C	300

GRANT 와 REVOKE

▶ GRANT에 있는 WITH GRANT OPTION 은

권한을 부여받은 사람이 해당 권한을 또다른 사람에게 부여할 수 있는 권한을 의미합니다.



TAB1

COL1	COL2
A	100
B	200
C	300

GRANT 와 REVOKE

▶ REVOKE 는 권한을 다시 회수하는 문법입니다.

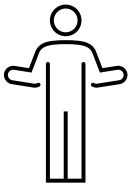
revoke 회수할권한 [on 회수할객체] from 회수당할계정

```
REVOKE SELECT ON USER_B.TAB1 FROM USER_A ;
```

권한이 취소되었습니다.

NO! 너는 SELECT 할 권한이 없어 .

내가 USRE_B의 TAB1 테이블을
조회하고 변경할 수 있을까?



USER_A

```
SELECT *  
FROM USER_B.TAB1 ;
```

질의 결과 x

SQL | 실행 중:SELECT * FROM C##USER_B,TAB1(0초)

ORA-00942: 테이블 또는 뷰가 존재하지 않습니다
00942, 00000 - "table or view does not exist"



USER_B

TAB1

COL1	COL2
A	100
B	200
C	300

GRANT 와 REVOKE 예시

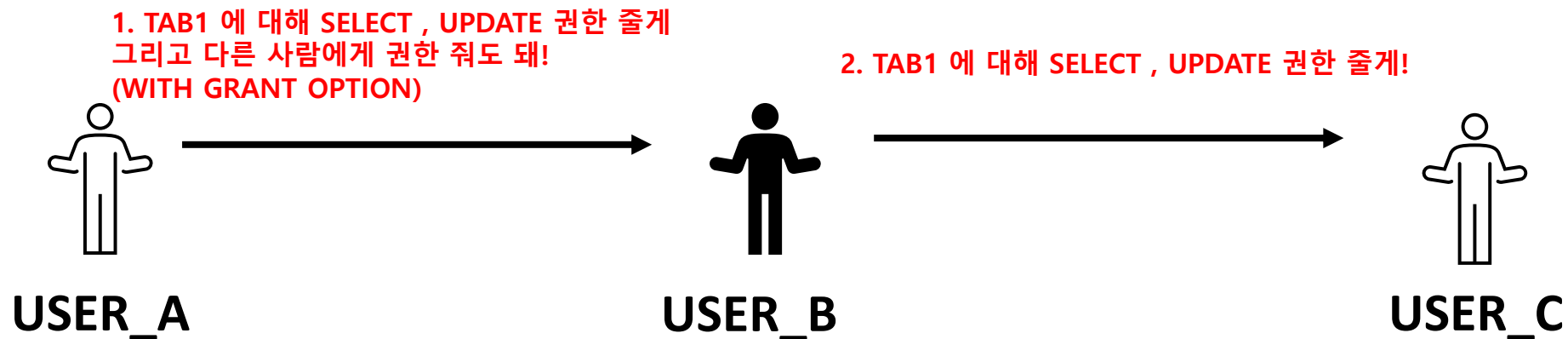
USER_A 가 TAB1 테이블을 가지고 있습니다.
그리곤 아래와 같이 다른 사용자(USER_B , USER_C) 들에게 권한을 부여합니다.

USER_A : GRANT SELECT , UPDATE ON TAB1 TO USER_B WITH GRANT OPTION ;

USER_B : GRANT SELECT , UPDATE ON USER_B.TAB1 TO USER_C ;

USER_A : REVOKE UPDATE ON TAB1 FROM USER_B ;

USER_A : REVOKE SELECT ON TAB1 FROM USER_B CASCADE ;



GRANT 와 REVOKE 예시

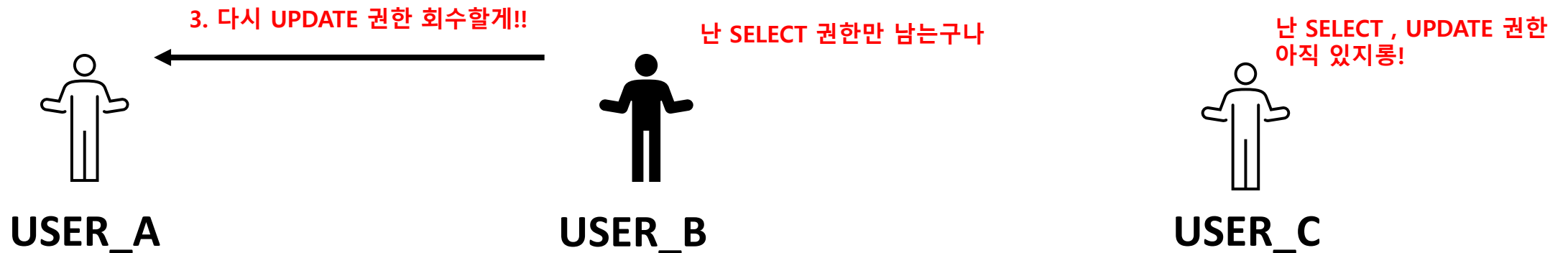
USER_A 가 TAB1 테이블을 가지고 있습니다.
그리곤 아래와 같이 다른 사용자(USER_B , USER_C) 들에게 권한을 부여합니다.

USER_A : GRANT SELECT , UPDATE ON TAB1 TO USER_B WITH GRANT OPTION ;

USER_B : GRANT SELECT , UPDATE ON USER_B.TAB1 TO USER_C ;

USER_A : REVOKE UPDATE ON TAB1 FROM USER_B ;

USER_A : REVOKE SELECT ON TAB1 FROM USER_B CASCADE ;



GRANT 와 REVOKE 예시

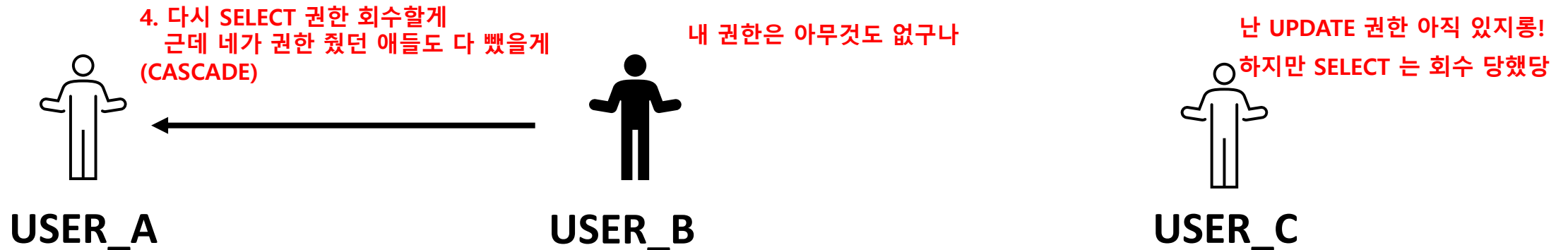
USER_A 가 TAB1 테이블을 가지고 있습니다.
그리곤 아래와 같이 다른 사용자(USER_B , USER_C) 들에게 권한을 부여합니다.

USER_A : GRANT SELECT , UPDATE ON TAB1 TO USER_B WITH GRANT OPTION ;

USER_B : GRANT SELECT , UPDATE ON USER_B.TAB1 TO USER_C ;

USER_A : REVOKE UPDATE ON TAB1 FROM USER_B ;

USER_A : REVOKE SELECT ON TAB1 FROM USER_B CASCADE ;



ROLE 이란?

ROLE 은 여러 권한을 한번에 부여할 수 있도록 묶어놓은 개념 입니다.


ROLE	CONNECT		RESOURCE	
	ALTER SESSION		CREATE CLUSTER	
	CREATE SESSION		CREATE PROCEDURE	
	CREATE SYNONYM		CREATE TABLE	
	CREATE TABLE		CREATE TREIGGER	
	CREATE VIEW		CREATE	

```
GRANT CONNECT , RESOURCE TO USER_A ;
```


28. 절차형 SQL (PL/SQL)

123 , 124(그냥암기) , 125 , 126, 127

SQL도 자바나 파이썬 처럼 **순서대로(절차대로) 프로그래밍**을 하고 **모듈화(=함수화)** 시킬 수 있으며 PL/SQL로 처리 가능합니다.



시작 부분 (필수)

입력 파라미터 지정

프로시저 끝 부분 (필수)

프로시저 실행

절차형SQL (PL/SQL 이란?)

프로시저 외에도 사용자정의함수(USER DEFINED FUNCTION) 나 트리거(TRIGGER) 도 생성가능

사용자정의함수의 경우 프로시저와

거의 동일하며, 출력(리턴)값이 존재해야 한다는

차이가 있습니다.

(지금까지 쓰던 단일행함수 등도 이런식으로 구현)

```
CREATE OR REPLACE Function CHECK_EMP (emp_id in varchar2) return VARCHAR2
IS v_return varchar2(10) := 'ok';
    cnt number := 0 ;
BEGIN
SELECT COUNT(*) into cnt FROM 직원 WHERE 직원ID = emp_id ;
if cnt > 0 then UPDATE 직원 SET 연봉 = 연봉 + 1000 WHERE 직원ID = emp_id ;
    v_return := 'ok' ;
else v_return := 'no' ;
end if;

RETURN v_return;

END;
```

리턴할 타입 지정

결과 리턴

절차형SQL (PL/SQL 이란?)

프로시저 외에도 **사용자정의함수**(USER DEFINED FUNCTION) 나 **트리거**(TRIGGER) 도 생성가능

트리거의 경우 특정 테이블에 DML 할 때

DB에서 자동으로 실행되도록 하는 모듈입니다.

보통 DML 할 때 에러가 나는 경우는 트리거가 자동실행되어

무결성 및 일관성을 체크한 것입니다.

(사용자가 직접 실행할 수 없으며 트랜잭션 제어도 불가능!)

```
CREATE OR REPLACE Trigger TRIGGER_TEST
AFTER INSERT ON 직원ID FOR EACH ROW
--
DECLARE ..
--
BEGIN ..
  //SQL 문장 ..
--
END; /
```

자동 실행될 시점

프로시저와 트리거의 차이 정리

프로시저

프로시저 생성은 CREATE PROCEDURE ..

실행은 EXEC 프로시저명 ..

내부에 트랜잭션제어어(COMMIT , ROLLBACK) 사용 가능

트리거

트리거 생성은 CREATE TRIGGER ..

실행은 DB 가 특정 조건때 알아서 실행

내부에 트랜잭션제어어(COMMIT , ROLLBACK) 사용 불가

해당 창작물의 허가되지 않은 재배포 , 무단 복사 및 무단 게시를 엄격히 금지합니다.
반드시 저작자에게 문의한 후 허가를 받아 사용해주세요.

(자료 중 빨간글씨로 작성된 숫자는 [SQL 자격검정 실전문제 , 한국데이터산업진흥원] 책의 문항을 의미하며 해당 개념으로 관련된 문항이라고 해석을 하시면 됩니다.)

SQLD 특강

21 ~ 28 SQL 활용 종료 PART2

강사 강태우

