**#SQL - 순서, 집합지향에 더해진 절차지향**

* 프로그래밍을 할 때, 특정 순서를 가진 데이터를 다루는 일이 자주 있다.
* 관계 모델의 이론 때문에 SQL은 전통적으로 순번을 다루는 기능을 가지고 있지 않다.
* 실질적으로 의미가 없는 순번은 엔티티의 속성으로 보지 않으므로, 테이블의 레코드를 순서 없게 정의하기 때문이다.
* 하지만 실제로는 레코드에 순번을 붙혀야 하는 경우가 많다.
* 이러한 요구에 따라, 최근 SQL은 순서와 순번을 다루기 위한 기능을 추가 하고 있다.
* 예를 들어, 시퀀스 객체 또는 ID 필드와 같은 순번을 붙일 수 있는 기능을 비롯해, 윈도우 함수 역시 이러한 요구에 대응한 기능이다.

1. **레코드에 순번 붙이기**
2. **기본키가 한 개의 필드일 경우**

* **ROW\_NUMBER() OVER (ORDER BY col)**

1. -- 체중테이블 정의
2. create table Weights
3. (student\_id char(4) primary key,
4. weight integer);
5. insert into Weights values ('A100', 50);
6. insert into Weights values ('A101', 55);
7. insert into Weights values ('A124', 55);
8. insert into Weights values ('B343', 60);
9. insert into Weights values ('B346', 72);
10. insert into Weights values ('C563', 72);
11. insert into Weights values ('C345', 72);
12. -- 윈도우 함수 사용
13. -- 학생 ID를 오름차순으로 순번 정의
14. SELECT student\_id, ROW\_NUMBER() OVER (ORDER BY student\_id) AS seq
15. FROM Weights;
16. -- 상관 서브쿼리(내부쿼리에서 외부 쿼리의 테이블을 참조)를 사용
17. -- MySQL처럼 ROW\_NUMBER 함수를 사용할 수 없는 환경에서는 상관서브쿼리를 사용해야 한다.
18. -- 이 서브쿼리는 재귀 집합을 만들고 요소 수를 COUNT 함수로 센다.
19. -- 기본키 student\_id를 비교 사용하므로 재귀집합의 요소가 한 개씩 증가한다.
20. -- 순번을 생성할 때 자주 사용하는 트릭이다.
21. SELECT student\_id,
22. (SELECT COUNT(\*)
23. FROM Weights W2
24. WHERE W2.student\_id <= W1.student\_id) AS seq
25. FROM Weights W1;
26. -- 두 가지 방법은 기능적으로는 동일하지만, 성능 측면에서 윈도우 함수를 사용하는 편이 좋다.
27. -- 윈도우 함수에서는 스캔 횟수가 1회.
28. -- 또 인덱스 온리 스캔을 사용하므로 테이블에 직접적인 접근을 회피한다.
29. -- 한편 상관 서브쿼리를 사용하는 방법에서는 2회(w1, w2)의 스캔이 실행된다.
30. **기본키가 여러 개의 필드일 경우**

* **ROW\_NUMBER() OVER(ORDER BY col1, col2)**

1. -- 체중테이블2 정의
2. CREATE TABLE Weights2
3. (class INTEGER NOT NULL,
4. student\_id CHAR(4) NOT NULL,
5. weight INTEGER NOT NULL,
6. PRIMARY KEY(class, student\_id));
8. INSERT INTO Weights2 VALUES(1, '100', 50);
9. INSERT INTO Weights2 VALUES(1, '101', 55);
10. INSERT INTO Weights2 VALUES(1, '102', 56);
11. INSERT INTO Weights2 VALUES(2, '100', 60);
12. INSERT INTO Weights2 VALUES(2, '101', 72);
13. INSERT INTO Weights2 VALUES(2, '102', 73);
14. INSERT INTO Weights2 VALUES(2, '103', 73);
15. -- 윈도우 함수 사용
16. -- ROW\_NUMBER를 사용하는 경우는 머리를 굴릴 필요가 전혀 없습니다.
17. -- ORDER BY의 키에 필드를 추가하기만 하면 됩니다.
18. SELECT class, student\_id, weight,
19. ROW\_NUMBER() OVER (ORDER BY class, student\_id) AS seq
20. FROM Weights2;
21. -- 상관 서브쿼리를 이용
22. -- 다중 필드 비교를 사용
23. -- 다중 필드 비교: 이름 그대로 복합적인 필드를 하나의 값으로 연결하고 한꺼번에 비교하는 기능
24. -- 이 방법의 장점은 자료형을 원하는대로 지정할 수 있다는 것 이다.
25. -- 숫자와 문자열, 문자열과 숫자라도 가능하다.
26. -- 암묵적인 자료형 변환도 발행하지 않으므로 기본키 인덱스도 사용 가능, 필드 개수 제한 없다.
27. -- 다중 필드 비교 사용 불가 : 오라클, ...
28. SELECT class, student\_id, weight,
29. (SELECT COUNT (\*)
30. FROM Weights2 W2
31. WHERE (W2.class, W2.student\_id) <= (W1.class, W1.student\_id) ) AS seq
32. FROM Weights2 W1;

**3) 그룹마다 순번을 붙이는 경우**

**- ROW\_NUMBER() OVER (PARTITION BY col1 ORDER BY col2)**

1. -- 그룹마다 순번을 붙이는 경우
2. -- 테이블을 그룹으로 나누고 그룹마다 내부 레코드에 순번을 붙이는 것이다.
3. -- 윈도우 함수 => PARTITION BY
4. SELECT class, student\_id,
5. ROW\_NUMBER() OVER (PARTITION BY class ORDER BY student\_id) AS seq
6. FROM Weights2;
7. -- 상관쿼리
8. SELECT class, student\_id,
9. (SELECT COUNT(\*)
10. FROM Weights2 W2
11. WHERE W2.class = W1.class
12. AND W2.student\_id <= W1.student\_id) AS seq
13. FROM Weights2 W1;
14. **순번과 갱신**

* **SET구문에 위의 SELECT구문 사용**

1. -- 순번과 갱신
2. -- 검색이 아니라 갱신에서 순번을 매기는 방법을 알아본다.
3. -- 순번을 갱신하는 UPDATE 구문을 만들어본다.
4. -- 체중테이블3 정의
5. CREATE TABLE Weights3
6. (class INTEGER NOT NULL,
7. student\_id INTEGER NOT NULL,
8. weight INTEGER NOT NULL,
9. seq INTEGER NULL,
10. PRIMARY KEY(class, student\_id) );
11. INSERT INTO Weights3 VALUES(1,'100', 50, null);
12. INSERT INTO Weights3 VALUES(1,'101', 55, null);
13. INSERT INTO Weights3 VALUES(1,'102', 56, null);
14. INSERT INTO Weights3 VALUES(2,'100', 60, null);
15. INSERT INTO Weights3 VALUES(2,'101', 72, null);
16. INSERT INTO Weights3 VALUES(2,'102', 73, null);
17. INSERT INTO Weights3 VALUES(2,'103', 73, null);
18. -- 윈도우 함수 사용
19. -- 기본적으로 앞서 본 순번 할당 쿼리를 SET 구로 넣는 방법을 생각할 수 있다.
20. UPDATE Weights3
21. SET seq = (SELECT seq
22. FROM (SELECT class, student\_id,
23. ROW\_NUMBER()
24. OVER(PARTITION BY class
25. ORDER BY student\_id) AS seq
26. FROM Weights3) SeqTbl
27. WHERE Weights3.class = SeqTbl.class
28. AND Weights3.student\_id = SeqTbl.student\_id);
29. -- 상관쿼리
30. -- 그냥 넣어주면 된다. (MySQL은 안됌. 갱신시 자기참조 불가)
31. UPDATE Weights3
32. SET seq = (SELECT COUNT(\*)
33. FROM Weights3 W2
34. WHERE W2.class =Weights3.class
35. AND W2.student\_id <= Weights3.student\_id);
36. **레코드에 순번 붙이기 응용**

* 테이블의 레코드에 순번을 붙일 수 있다면, 자연 수열(순번)의 성질을 활용한 다양한 테크닉을 사용할 수 있다.
* ‘연속성’과 ‘유일성’을 사용해본다.

1. **중앙값 구하기**

* 중앙값이란 숫자를 정렬하고 양쪽 끝에서부터 수를 세는 경우 정중앙에 오는 값 이다.
* 단순 평균과 다르게 아웃라이어에 영향을 받지 않는다는 장점이 있다.
* 데이터의 개수가 홀수라면 중앙값을 그냥 사용한다.
* 짝수라면 중앙에 있는 두 개의 값의 평균을 사용한다.
* **중심을 향해서**

. 반드시 ROW\_NUMBER 함수를 사용해야 한다. RANK 또는 DENSE\_RANK는 순번의 연속성 벗어날 수 있다.

. ORDER BY의 정렬 키에 weight 필드뿐만 아니라 기본키인 student\_id도 포함해야한다. 동일 weight인 경우, 연속성을 유지하며 정렬될 수 있도록 정렬기준을 줘야 한다.

-- SQL에서 자연수의 특징을 활용하면 '양쪽 끝부터 숫자 세기'를 할 수 있다.

-- 홀수의 경우에는 hi = lo

-- 짝수의 경우에는 hi = lo+1 과 hi = lo-1의 두 개가 존재한다.

SELECT AVG(weight) AS median

FROM (SELECT weight,

ROW\_NUMBER() OVER (ORDER BY weight ASC, student\_id ASC) AS hi,

ROW\_NUMBER() OVER (ORDER BY weight DESC, student\_id DESC) AS lo

FROM Weights) TMP

WHERE hi IN (lo, lo+1, lo-1);

* **2빼기 1은 1**

. ROW\_NUMBER를 사용하는 경우 코드의 가독성은 좋지만 가장 좋은 성능을 내지는 않는다.

SELECT AVG(weight)

FROM (SELECT weight,

2 \* ROW\_NUMBER() OVER(ORDER BY weight) - COUNT(\*) OVER() AS diff

FROM Weights) TMP

WHERE diff BETWEEN 0 AND 2;

. 해당 연산을 처리한 경우, diff 값이 0과 2사이의 값인 weight의 평균값이 중간값을 갖는다.

. 정렬을 한번만 사용하므로 성능이 좋다.

. COUNT 함수의 OVER구문에 ORDER BY가 없으므로, 옵티마이저는 단순하게 ROW\_NUMBER 함수의 OVER 구문의 ORDER BY만 정렬로 계획하기 때문이다.

1. **순번을 사용한 테이블 분할**

* 집합 지향적 방법 – ‘집합의 경계선’

. 전통적인 SQL에서는 레코드 단위가 아니라 집합 단위로 생각한다.

-- 집합 지향 - 집합의 경계선

SELECT (N1.num +1) AS gap\_start, '~', (MIN(N2.num) - 1) AS gap\_end

FROM Numbers N1 INNER JOIN Numbers N2

ON N2.num > N1.num

GROUP BY N1.num

HAVING (N1.num+1) < MIN(N2.num);

* 절차 지향적 방법 – ‘다음 레코드’와 비교

. 현재 레코드와 다음 레코드를 비교해서 차이가 1이 아니면 => **diff <> 1**

. Numbers 테이블에 한 번만 접근이 발생하고 윈도우 함수에서 정렬이 실행된다.

. 결합을 사용하지 않으므로 성능이 굉장히 안정적이다.

-- next\_num : MAX(num) OVER (ORDER BY num ROWS BETWEEN 1 FOLLOWING AND 1 FOLLOWING)

SELECT num+1 AS gap\_start, '~' , (num+diff-1) AS gap\_end

FROM (SELECT num,

MAX(num) OVER (ORDER BY num ROWS BETWEEN 1 FOLLOWING

AND 1 FOLLOWING) - num AS diff

FROM Numbers) TMP

WHERE diff <>1; -- (다음수 - 현재수)가 1이 아니라면 -> gap 발생한 경우

1. **테이블에 존재하는 시퀀스 구하기**
2. **시퀀스 객체, IDENTITY 필드, 채번 테이블**
3. **시퀀스 객체**
4. **IDENTITY 필드**
5. **채번 테이블**