**데이터를 자유자재로! SQL 기초 2**

01 집합연산자 & 계층형질의

1. Standard SQL

관계형 대수: 관계형 데이터베이스에서 원하는 정보를 유도하기 위한 기본 연산 집합

\*일반 집합 연산

합집합: union

교집합: intersect

차집합: except

카디션 프로덕트(cartesian product)(순열?): cross join

\*순수 관계 연산

셀렉션: 특정 행들 출력(where)

프로젝션: 특정 컬럼들 출력(selection)

조인(세타,동등,자연,외부 조인):

디비전: 조건 검색?(개념만 존재하고 실제 연산은 없음)

2. 집합 연산자

두 개 이상의 테이블에서 조인을 사용하지 않고 연관된 데이터를 조회하는 방법 중 하나

테이블에서 select한 컬럼의 수와 각 컬럼의 데이터타입이 테이블 간 상호 호환 가능해야 한다.

union, union all, intersect, except

union

데이터 연결 및 정렬, 중복 제거

select \* from A

union

select \* from b

union all

데이터 정렬x, 중복제거x

단순히 연결시켜줌

intersect

oracle/maria DB는 지원하지만

MySQL에서는 지원하지 않아서 inner join을 활용해야 함

select a,b from Alpha

intersect

select a,b from beta;

->

select a,b from Alpha inner join beta on Alpha.a= beta.a

except(minus)

oracle DB에서는 지원(minus), Maria DB는 10.3버전부터 지원

MySQL에서는 지원하지 않아서 join을 활용해야 함

select a,b from Alpha

except

select a,b from beta;

->

Select a,b from A left join B

On A.a = B.b

Where B.b is null

3. 계층형 질의

테이블에 계층형 데이터가 존재하는 경우 데이터를 조회하기 위해 사용하는 것

계층형: 동일 테이블에 계층적으로 상위와 하위 데이터가 포함되어 있는 데이터

-Oracle

select level,자식 컬럼, 부모 컬럼, 원하는 컬럼

from 테이블

start with 부모 컬럼 is null (부모 컬럼이 Null인 행이 root가 됨)

connect by prior 부모 컬럼 = 자식 컬럼 (상위 데이터와 하위 데이터의 연결 방식)

\*lpad 함수(왼쪽에 n자리 공백 추가) 응용

select level, lpad(‘ ’,4\*(level-1))||사원번호, 관리자

from ~

level: 검색항목의 깊이(루트는 1)

connect\_by\_root: 현재 전개할 데이터의 루트 데이터 값 표시

connect\_by\_isleaf: 최하위 데이터인지 표시(0 or 1)

sys\_connect\_by\_path(컬럼명, 구분자): 루트 데이터부터 현재까지 경로 표시

-SQL server

-MySQL/MariaDB

CTE로 재귀호출 가능

크게 형태가 안변함

CTE(common table expression)

WITH RECURSIVE CTE(member\_id, manager\_id, lvl)

AS (

SELECT member\_id, manager\_id, 0 AS lvl

FROM MEMBER

WHERE manager\_id IS NULL

UNION ALL

SELECT a.member\_id, a.manager\_id, b.lvl + 1

FROM MEMBER a

JOIN CTE AS b

ON a.manager\_id = b.member\_id

)

SELECT member\_id, manager\_id, lvl

FROM CTE

ORDER BY member\_id, lvl;

02 Join 심화

EQUI Join: 두 개의 테이블 간에 서로 정확하게 일치하는 경우를 활용하느 조인(=)

Non EQUI Join: 두 개의 테이블 간에 서로 정확하게 일치하지 않는 경우를 활용하느 조인(>,>=,<=,<,between)

From절 join 형태

1. (inner) join

select \* from user a join class b

on a.class\_id=b.id;

테이블 3개 이상일 때

select request\_id, student\_name, class\_name from CLASS\_REQUEST  
join STUDENT USING (STUDENT\_ID)  
JOIN CLASS USING (class\_id)  
order by request\_id;

2. using: 같은 이름을 가진 컬럼들 중 원하는 컬럼에 대해서만 선택적으로 등가 조인 가능(SQL Server에서는 지원 X)

select \* from 테이블1 join 테이블2

using(칼럼);

3. natral join: 두 테이블 간의 동일한 이름을 갖는 모든 컬럼들에 대해 등가 조인을 수행

select \* from 테이블1 natural join 테이블2;

(cross) join: join 조건이 없는 경우 생길 수 있는 모든 데이터의 조합을 조회

select \* from person

(cross) join public\_transport;

join에 on이 있으면 inner join on이 없으면 cross join

outer join: 두 개의 테이블 간에 교집합을 조회하고 한쪽 테이블에만 있는 데이터도 포함시켜서 조회, 빈 곳은 NULL값으로 출력

where조건절에서 한쪽에만 있는 데이터를 포함시킬 테이블 쪽으로(+)를 위치

select \* from user,class

where user.class\_ID (+) = class.class\_ID

동일한 결과

select \* from user left (outer) join class

on user.class\_ID = class.class\_ID

\*right (outer) join도 있지만 통상적으로 left를 많이 씀

full outer join: left join + right join (교집합+왼쪽만+오른쪽만)

select \* from class full outer join user

on user.class\_ID = class.class\_ID;

oracle에서 full outer join 지원하지만

Maira DB, MySQL에서는 union(중복 데이터를 제거)를 이용해서 사용 가능

(left join) union (right join)

정리

inner join

outer join

cross join

\*where문과 join문은 따로임

셀프 조인: 동일 테이블 사이의 조인이르모 식별을 위해 별칭 필수

select a.칼럼, b.칼럼

from 테이블1 a, 테이블1 b

where a.칼럼2 = b.칼럼1

03 서브쿼리 심화

1. 동작하는 방식에 따른 서브쿼리 분류

메인쿼리의 컬럼이 포한되는지에 따라 구분

연관 서브쿼리: 메인쿼리의 컬럼이 서브쿼리에 포함되며, 메인쿼리의 컬럼은 서브쿼리에 특정 조건으로 사용(서브쿼리 자체만으로 실행 불가능)

select \* from employee a

where salary > (

select avg(salary) from employee b

where b.id = a.id);

비연관 서브쿼리: 메인쿼리의 컬럼이 서브쿼리에 포함되지 않으며, 주로 메인 쿼리에 특정한 값을 제공할 때 사용된다. (서브쿼리 자체만으로 실행 가능)

select avg(salary) from employee

where id=(

select id

from employee

where name=’elice’);

2. 반환되는 데이터 형태에 따른 서브쿼리 분류

단일 행 서브쿼리: 서브쿼리의 결과가 한 개의 행을 반환하며, 비교 연산자(=,<,>,<=,>=)와 같이 사용

다중 행 서브쿼리: 서브쿼리의 결과가 두 개 이상 행을 반환, 연산자(IN, ALL, ANY, EXISTS)와 같이 사용

SELECT NAME

FROM EMPLOYEE A

WHERE EXISTS (

SELECT ID FROM EMPLOYEE B

WHERE

B.SALARY >= 10000 AND

A.DEPARTMENT\_ID = B.DEPARTMENT\_ID);

\*exists 서브쿼리를 이용할때는 select 은 중요하지 않음, 존재 유무만 판단하기 떄문에 from where이 중요

다중 컬럼 서브쿼리: 서브쿼리의 결과가 여러 개의 컬럼을 반환하며, 메인쿼리의 조건과 동시에 비교된다.

SELECT NAME, SALARY FROM EMPLOYEE

WHERE (DEPARTMENT\_ID, SALARY) IN (SELECT DEPARTMENT\_ID, MAX(SALARY)

FROM EMPLOYEE GROUP BY DEPARTMENT\_ID);

스칼라 서브쿼리: 하나의 속성을 가지면서, 하나의 행만을 반환하는 쿼리이다.

그리고 이는 SELECT, WHERE, HAVING 절 등에서 사용할 수 있다.

SELECT NAME, (SELECT

COUNT(\*) FROM EMPLOYEE E

WHERE E.DEPARTMENT\_ID = D.ID)

FROM DEPARTMENT D;

DUAL

스칼라 서브쿼리 FROM을 사용한 특정 테이블 참조없이 쿼리를 작성하는 경우도 있다.

전체 임직원중에 팀장직급이 차지하는 비율을 출력

SELECT

(SELECT COUNT(\*) FROM EMPLOYEE WHERE DEPARTMENT\_ID = 1) /

(SELECT COUNT(\*) FROM EMPLOYEE)

AS DEVELOPER\_RATIO

(FROM DUAL;) <-Oracle에서는 의무 작성, MySQL, MariaDB에서는 생략 가능

뷰는 다른테이블에서 파생된 테이블이다. 물리적으로 데이터가 저장되는 것이 아니라, 논리적으로만 존재하며 뷰를 사용한질의시에는 DBMS에서 뷰정의에 따라 질의를 재작성하여 수행한다

장점

독립성

편리성

보안성

특징

생성된뷰는또다른뷰를생성하는데사용될수있다

• 뷰의 정의는 변경할 수 없으며, 삭제후 재생성이 필요하다(delete와 create로 가능 -> replace)

• 뷰를 통한 갱신에는 제약이 따른다. 갱신을 위해서는 기본적으로 원천테이블의 기본키가 포함되어야한다

• 원천이 되는 테이블이나 뷰가 삭제되면 이를 기반으로하는 뷰도 함께 삭제된다.

CREATE [OR REPLACE] VIEW view\_name AS [같은 이름 존재시 무시하고 만듬]

SELECT column1, column2, …

FROM table\_name

WHERE condition;

04 그룹 함수 & 윈도우 함수

데이터 분석 개요

윈도우 함수: 순위, 집계 등 행과 행 사이의 관계를 정의하는 함수

Over 구문을 필수로 한다.

Select window\_function (arguments)

Over (partition by 컬럼 order by절 windowing절) from 테이블 명

arguments: 인수

partition by 전체 집합에 대해 소그룹으로 나누는 기준

order by 소그룹에 대한 정렬 기준

windowing 행에 대한 범위 기준

rows: 물리적 단위로 행의 집합을 지정

unbounded preceding: 윈도우의 시작위치가 첫번째 행

unbounded following: 윈도우의 마지막 위치가 마지막 행

current row: 윈도우의 시작위치가 현재 행

순위함수

Rank(): 동일 순위에 대해 다건으로 취급

Dense\_rank(): 동일 순위에 대해 단건으로 취급

row\_number(): 동일 순위 없음

SELECT ID, NAME, SALARY,

RANK() OVER (ORDER BY SALARY DESC) RANK,

DENSE\_RANK() OVER (ORDER BY SALARY DESC) DENSED\_RANKING,

ROW\_NUMBER() OVER (ORDER BY SALARY DESC) ROW\_NUMBER

FROM EMPLOYEE;

집계 함수(윈도우 함수에 포함)

Avg,Sum,max,min

SELECT ID, NAME, SALARY,

AVG(SALARY) OVER (PARTITION BY DEPARTMENT\_ID) DEPARTMENT\_AVG

FROM EMPLOYEE;

그룹 함수

그룹 내 행 순서 함수

first\_value: 가장 먼저 나온 값

last\_value: 가장 나중에 나온 값

lag(컬럼,x)): 이전 x번째 행

lead(컬럼,x): 이후 x번째 행

SELECT

ID, DEPARTMENT\_ID, NAME, SALARY,

FIRST\_VALUE(SALARY) OVER(PARTITION BY DEPARTMENT\_ID ORDER BY SALARY

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) DEPARTMENT\_MIN\_SALARY ,

LAST\_VALUE(SALARY) OVER(PARTITION BY DEPARTMENT\_ID ORDER BY SALARY

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) DEPARTMENT\_MAX\_SALARY

FROM EMPLOYEE

ORDER BY ID;

SELECT ID, NAME, SALARY,

LAG(NAME, 1) OVER(ORDER BY ID) PREV\_EMPLOYEE\_NAME,

LEAD(NAME, 1) OVER(ORDER BY ID) AFTER\_EMPLOYEE\_NAME

FROM EMPLOYEE;

그룹 내 비율 함수

1. RATIO\_TO\_REPORT: 파티션내 전체 SUM에 대한 비율을 구한다(MariaDB는 안됨)

2. PERCENT\_RANK: 파티션내 순위를 백분율로 구한다(0이상 1이하 값 반환)

3. CUME\_DIST: 파티션내 현재행보다 작거나 같은건들의 누적백분율로 구한다

4. NTILE(n): 파티션내 행들을 N등분한 결과를 구한다.(딱 등분이 안되면 1등부터 다시 채움(ex)8/3=>3,3,2))

SELECT ID, NAME, SALARY,

SUM(SALARY) OVER() TOTAL\_SALARY,

RATIO\_TO\_REPORT(SALARY) OVER() RATIO\_TO\_REPORT

FROM EMPLOYEE;

\*MariaDB에서는 RATIO\_TO\_REPORT가 제공되지 않음 -> (SALARY / SUM(SALARY) OVER())

2,3

SELECT ID, NAME, SALARY,

PERCENT\_RANK() OVER(ORDER BY SALARY DESC) AS PERCENT\_RANK,

ROUND(CUME\_DIST() OVER(ORDER BY SALARY DESC), 4) AS CUME\_DIST

FROM EMPLOYEE;

4.

SELECT ID, NAME, SALARY,

NTILE(3) OVER(ORDER BY SALARY DESC) NTILE

FROM EMPLOYEE;

그룹함수

데이터를 통계내기 위해서는, 전체 데이터에 대한 통계는 물론이고 데이터 일부에 대한 소계, 중계 또한 필요하다.

각 레벨 별 SQL을 UNION문으로 묶어 작성할 수도 있으나 ORACLE DB에서는이러한 통계데이터 위한 몇 가지 함수를 제공한다

Group by: 일반집계함수의 over(partition by로 대체 가능) ,but 실제로는 group by가 더 대중적

roll up(뒤에 인수부터 말아 올라간다)(순서에 영향 O): 그룹화하는 컬럼에 대한 부분적인 통계를 제공해준다

oracle에서는 group by roll up(~), MariaDB에서는 group by ~ with rollup;

cube: ROLLUP 함수에서 제공하는 결과를 포함해서, CUBE 함수에서는 그룹화하는 컬럼에 대해

결합가능한 모든 경우의 수에 대해 다차원집계를 생성한다

(rollup함수의 모든 경우를 보여줌)(인수 순서에 영향X)

\*But MariaDB에서는 지원을 안해서 ~ with roll up(~) 여러 개를 union해야함

Grouping sets: 명시된 컬럼에 대해 개별통계를 생성한다

각 컬럼에 대해 GROUP BY로 생성한 통계를 모두 UNION ALL한 결과와 동일하다.

\*But MariaDB에서는 지원을 안해서 각 Group by를 한 것을 union all해야함