SQL – RANKING AND WINDOWING

CIS-673, LECTURE#09

BY RAJ PATIL

2 AGGREGATE (VS) ANALYTIC FUNCTIONS

- **aggregate function:** aggregates data from several rows into a single result row.
- **GROUP BY clause:** applies aggregate functions to subsets of rows.
- **Analytic functions:** also operate on subsets of rows, similar to aggregate functions in GROUP BY queries, but they do not reduce the number of rows returned by the query.

3 ANALYTIC FUNCTION SYNTAX

- analytic_function([arguments]) OVER (analytic_clause)
- Use OVER analytic_clause to indicate that the function operates on a query result set.

- **analytic_clause:** is computed after the FROM, WHERE, GROUP BY, and HAVING clauses, but before the final ORDER BY operation is performed.

- analytic functions are sometimes referred to as **window[ing] functions**.

4 ANALYTIC FUNCTION SYNTAX

- The analytic_clause breaks down into the following optional elements.
  - partition_clause
  - order_by_clause
  - windowing_clause

5 PARTITION_CLAUSE

- partition_clause divides the result set into partitions, or groups, of data.
- The operation of the analytic function is restricted to the boundary imposed by these partitions, similar to the way a GROUP BY clause affects the action of an aggregate function.
- If the partition_clause is omitted, the whole result set is treated as a single partition.

6 ORDER_BY_CLAUSE

- The order_by_clause is used to order rows within a partition.
- In analytic functions, the order of rows matters, as the results may vary as per the order.
- if an analytic function is sensitive to the order in a partition you should include an order_by_clause.

7 WINDOWING_CLAUSE

- partition_clause controls the window, or group of rows, the analytic operates on.
- windowing_clause gives some analytic functions a further degree of control over this window within the current partition (or whole result set if no partitioning clause is used.)
- The windowing_clause has two basic forms:
  - RANGE BETWEEN start_point AND end_point
  - ROWS BETWEEN start_point AND end_point

8 RANK - USING CORRELATED IN THE SELECT CLAUSE

```
select name, gpa, (1 + (select count(*)
                        from student_grades B
                        where B.GPA > A.GPA)
                ) as s_rank
from student_grades A
order by s_rank;
```

9 USING RANK FUNCTION

- RANK() : creates ranks based on values in specified column and order

- **Query:** Rank students based on their GPA

- **SQL:** select name, gpa, rank() over (order by GPA desc) as stu_rank

  from student_grades

  order by stu_rank;

- **SQL:** select name, gpa, rank() over (order by GPA desc nulls last) as stu_rank

  from student_grades

  order by stu_rank;

10 RANK AND PARTITION

- **Query:** For every department, rank students based on their GPA

- **SQL:** select name, dept, gpa, rank() over (partition by dept order by GPA desc nulls last) as stu_rank

from student_grades

order by dept, stu_rank;

Query: For every department, rank employees based on their salary

SQL: select empno, deptno, sal, rank() over (partition by deptno order by sal) as emp_rank

from emp

order by deptno, sal;

11 WINDOWING

- Windowing – used to compute MOVING info.
  - Perform or apply aggregate functions [sum(), min(), max()…]over a window
- Before sliding the window, the data can be 'Partitioned by' and/or 'ordered by' as per the requirement
- Keywords: Preceding, following, unbounded, current row, rows between <start, end>, rows <start>

12 WINDOWING [CONT…]

- Specify the start and end point of the window
  - Default <end> point of the window is the current row;
  - No default value for <start>; it has to be mentioned
- Unlike groupby clause, in windowing technique, the number of rows in the result remain the same.

13 STUDENT AND EMPLOYEE TABLES

```
create table student_grades(
    name varchar(10),
    dept varchar(20),
    gpa numeric(3,2)
);
insert into student_grades values('Kim','CIS',3.75);
insert into student_grades values('Sam','CIS',3.4);
insert into student_grades values('John','Phy',3.9);
insert into student_grades (name, dept)
values('martha','Phy');

select * from student_grades;
```

```
create table emp (
  empno    number(4) constraint pk_emp primary key,
  ename    varchar2(10),
  job      varchar2(9),
  mgr      number(4),
  hiredate date,
  sal      number(7,2),
  comm     number(7,2),
  deptno   number(2)
);
insert into emp values (7369,'SMITH','CLERK',7902,to_date('17-12-1980','dd-mm-yyyy'),800,null,20);
insert into emp values (7499,'ALLEN','SALESMAN',7698,to_date('20-2-1981','dd-mm-yyyy'),1600,300,30);
insert into emp values (7521,'WARD','SALESMAN',7698,to_date('22-2-1981','dd-mm-yyyy'),1250,500,30);
…
select * from emp;
```

14 SALES TABLE

```
create table sales(
  year int,
  country varchar(15),
  product varchar(15),
  quantity_sold int
);
```

```
insert into sales values(2000,'Canada','laptops',500);
insert into sales values(2001,'Canada','smartphones',350);
insert into sales values(2002,'Canada','tablets',200);
insert into sales values(2000,'Finland','laptops',1500);
insert into sales values(2001,'Finland','smartphones',10);
insert into sales values(2002,'Finland','tablets',100);
insert into sales values(2000,'UK','laptops',75);
insert into sales values(2001,'UK','smartphones',1200);
insert into sales values(2002,'UK','tablets',275);
insert into sales values(2000,'USA','laptops',175);
insert into sales values(2001,'USA','smartphones',150);
insert into sales values(2002,'USA','tablets',2500);
```

15 WINDOWING – PRECEDING AND CURRENT ROW

- **Query:** Order the data by country and product, and then find moving sum of quantity sold by considering 2 previous rows.
- **SQL:** select year, country, product, sum(quantity_sold) over (order by country, product rows 2 preceding) as sum_prof

  from sales;

//OR

- **SQL:** select year, country, product, sum(quantity_sold) over (order by country, product rows between 2 preceding and current row) as sum_prof

  from sales;

16 WINDOWING – UNBOUNDED, FOLLOWING

- **SQL:** select year, country, product, sum(quantity_sold) over (order by country, product rows unbounded preceding) as sum_prof

  from sales;

- **SQL:** select year, country, product, sum(quantity_sold) over (order by country, product rows between 1 preceding and 1 following) as sum_prof

  from sales;

17 FOLLOWING REQUIRES START POINT

- select year, country, product, sum(quantity_sold) over (partition by year order by country, product rows 2 preceding) as sum_prof

  from sales;

- //when using the 'following' keyword, the <start> point has to be mentioned. Below, the query will result in error
- select year, country, product,

  sum(quantity_sold) over (partition by year order by country, product rows 2 following) as sum_prof

  from sales;

18 RANGE (W.R.T.VALUE)

- **Row:** cover specific number of rows/tuples //w.r.t. row count
- **Range:** cover all tuples with particular value(s) //w.r.t. row value

- **Query:** find the moving sum of quantities sold for the current and its previous (1 preceding) year
- **SQL:** select year, country, product,

  sum(quantity_sold) over (order by year range 1 preceding) as sum_prof

  from sales;

//try running the same command using 'rows' keyword instead of 'range'

19 RANGE - EXAMPLES

- **Query:** find the moving sum of quantities sold by taking into account previous 2 years
- **SQL:** select year, country, product,

  sum(quantity_sold) over (order by year range between 2 preceding and current row) as sum_prof

  from sales;

- select year, country, product,
- sum(quantity_sold) over (order by year range between current row and unbounded following) as sum_prof
- from sales;

20 LAG AND LEAD

- LAG and Lead are analytic function. They provide access to more than one row of a table at the same time without a self join.

- Given a series of rows returned from a query and a position of the cursor,

  - LAG function is used to access data from a previous row..

  - LEAD function is used to return data from rows further down the result set.

FIRST_VALUE, LAST_VALUE

- The FIRST_VALUE returns the first result from an ordered set.
- The LAST_VALUE returns the last result from an ordered set