SE3DB3 TUTORIAL

Hongfeng Liang

Oct 24/27, 2016

Outline

- Aggregation
- Count
- AVG
- SUM
- MIN and MAX
- Grouping

Aggregation

Function Syntax	Function Usage
SUM([ALL | DISTINCT] expression)	The total of the (distinct) values in a numeric column/expression.
AVG([ALL | DISTINCT] expression)	The average of the (distinct) values in a numeric column/expression.
COUNT([ALL |DISTINCT] expression)	The number of (distinct) non-NULL values in a column/expression.
COUNT(*)	The number of selected rows.
MAX(expression)	The largest value in a column/expression.
MIN(expression)	The smallest value in a column/ expression.

Aggregation

- There are two rules that you must understand and follow when using aggregates:
  - Aggregate functions can be used in both the SELECT and HAVING clauses (the HAVING clause will be covered later).
  - Aggregate functions cannot be used in a WHERE clause

Count

- If a manager needs to know how many employees work in the organization, COUNT(*) can be used to produce this information.
- The COUNT(*) function counts all rows in a table.
- * would be used as the parameter in the function.

```
SELECT COUNT(*)
FROM employee;
COUNT(*)
--------
8
```

Count

- The result table for the COUNT(*) function is a single scalar value.
- Notice that the result table has a column heading that corresponds to the name of the aggregate function specified in the SELECT clause.
- The output column can be assigned a more meaningful column name as is shown in the revised query.

Count

- This is accomplished by using keyword “as” plus the desired column name after the aggregate function specification.
- It is better to use “as” since if you query from Java which will a headache

```
SELECT COUNT(*) as NumberOfEmployees
FROM employee;
NumberOfEmployees
-----------------
8
```

Count

- COUNT(*): count all the rows in a talbe.
- COUNT(column name): define a specific column to be counted.
- When using COUNT(column name), rows containing a NULL value in the specified column are omitted. In contrast the COUNT(*) will count each row regardless of NULL values.
- A NULL values stands for “unknown” or “unknowable” and must not be confused with a blank or zero.

AVG

- AVG function is used to compute the average value for a column.
- For example, the following query returns the average of the employee salaries.

```
SELECT AVG(emp_salary) as
AverageEmployeeSalary
FROM employee;
AverageEmployeeSalary
--------------------
$35,500
```

AVG

- What is the average salary offered for all employees?
- This question ask you to incorporate the concept of computing the average of the distinct salaries paid the organization.
- The same query with DISTINCT keyword in the aggregate function returns a different average.

```
SELECT SUM(emp_salary) as TotalSalary
FROM employee;
TotalSalary
-----------
$284,000
```

SUM

- The SUM function can compute the total of a specified table column.
- The SELECT statement shown here will return the total of the emp_salary column from the employee table.

```
SELECT SUM(emp_salary) as TotalSalary
FROM employee;
TotalSalary
-----------
$284,000
```

SUM

- If management is preparing a budget for various departments, you may be asked to write a query to compute the total salary for different departments.
- The query show here will compute the total emp_salary for employees assigned to department #7.

```
SELECT SUM(emp_salary) as
TotalSalaryDept7
FROM employee
WHERE emp_dpt_number = 7;
TotalSalaryDept7
----------------
$136,000
```

MIN and MAX

- The MIN function returns the smallest value stored in a data column.
- The MAX function returns the largest value stored in a data column.
- Unlike SUM and AVG, the MIN and MAX functions work with both numeric and character data columns(well order set)

MIN and MAX

- A query that uses the MIN and MAX functions to find the smallest and largest values stored in the emp_last_name column of the employee table.
- MIN will return the employee row where last name comes first(smallest) in the alphabet.
- MAX will return the emplyee row where last name comes last(largest) in the alphabet.

```
SELECT MIN(emp_last_name), MAX(emp_last_name)
FROM employee;
MIN(EMP_LAST_NAME) MAX(EMP_LAST_NAME)
------------------------- ----------------------
Amin                      Zhu
```

Grouping

- The power of aggregate functions is greater when combined with the GROUP BY clause.
- In fact, the GROUP BY clause is rarely used without an aggregate function.
- It is possible to use the GROUP BY clause without aggregates, but such a construction has very limited functionality, and could lead to a result table that is confusing or misleading.

Grouping

- If any aggregation is used, then each element of the SELECT list must be either:
  - Aggregated, or
  - An attribute on the GROUP BY list
- The attribute name used in a GROUP BY does not have to be listed in the SELECT clause; however, it must be a column name from one of the tables listed in the FROM clause.

Grouping

```
SELECT COUNT(*) as DepartmentCount
FROM employee
GROUP BY emp_dpt_number;
DepartmentCount
---------------
1
3
4
```

Grouping

- However, the revese is **NOT** true!

```
SELECT emp_dpt_number, COUNT(*) as
DepartmentCount
FROM employee;
```

Grouping

- To keep it simple, just remember the following:
  - If you have column name(s) AND Aggregate Function(s) in the SELECT caluse, then you MUST also have a GROUP BY clause.
  - The column name(s) in the SELECT clause MUST match column name(s) listed in the GROUP BY clause.

Grouping with Where clause

- The WHERE clause works to eliminates data table rows from consideration before any grouping takes place.
- The query show here produces an average hours worked result table for employees with a social security number that greater than 999-66-0000.

```
SELECT work_emp_ssn as SSN,
AVG(work_hours) as AverageHoursWorked
FROM assignment
WHERE work_emp_ssn > 999660000
GROUP BY work_emp_ssn;
SSN         AverageHoursWorked
--------- ------------------
999666666         19.5
999887777         20.5
999888888         21.5
```

Order by

- The ORDER BY clause allows you to specify how rows in a result table are sorted.
- The default ordering is from smallest to largest value(asc)
- When using order by desc, the ordering change to from largest to smallest value.

Order by example

```
SELECT emp_dpt_number as Department,
AVG(emp_salary) as AverageSalary
FROM employee
GROUP BY emp_dpt_number
ORDER BY AVG(emp_salary);
Department AverageSalary
---------- -------------
3                $31,000
7                $34,000
1                $55,000
```

Group by with Having clause

- The HAVING clause is used for aggregate functions in the same way that a WHERE clause is used for column names and expressions.
- The HAVING and WHERE clauses do the same thing, that is filter rows from inclusion in a result table based on a condition.
- a WHERE clause is used to filter rows BEFORE the GROUPING action.
- a HAVING clause filters rows AFTER the GROUPING action

Example

```
SELECT emp_dpt_number as Department,
AVG(emp_salary) as AverageSalary
FROM employee
GROUP BY emp_dpt_number
HAVING AVG(emp_salary) > 33000;
Department AverageSalary
---------- -------------
1                $55,000
7                $34,000
```

Combing WHERE with HAVING

```
SELECT emp_dpt_number as Department,
AVG(emp_salary) as AverageSalary
FROM employee
WHERE emp_dpt_number <> 1
GROUP BY emp_dpt_number
HAVING AVG(emp_salary) > 33000;
Department AverageSalary
---------- -------------
7                $34,000
```

Performing steps

- Conceptually, SQL performs the following steps in the query given above:
  - The WHERE clause filters rows that do not meet the condition *emp_dpt_number* <> 1.
  - The GROUP BY clause collects the surviving rows into one or more groups for each unique emp_dpt_number
  - The aggregate function calculates the average salary for each emp_dpt_number grouping.
  - The HAVING clause filters out the rows from the result table that do not meet the condition average salary greater than $33,000

Natural left outer join

A	B
a	1
b	2

R1

A	C
a	3
c	5

R2

A	B	C
a	1	3
b	2	null

Natural right outer join

A	B
a	1
b	2

R1

A	C
a	3
c	5

R2

A	B	C
a	1	3
c	null	5

Natural full outer join

A	B
a	1
b	2

R1

A	C
a	3
c	5

R2

A	B	C
a	1	3
b	2	null
c	null	5

Insert

A	B
a	1
b	2

R1

A	C
a	3
b	5

R2

R2.A is a foreign key references to R1.A
Insertions which introduce nonexistent values must be rejected. E.g.
Insert into R2 values(“c”,6)
Will be rejected since value c is not appeared in R1.A

Update&Delete

A	B
a	1
b	2

R1

A	C
a	3
b	5

R2

R2.A is a foreign key references to R1.A
Any update and delete operations to R2 will be accepted.
However, update or delete to R1.A will be rejected by default

Cascade

A	B
a	1
b	2

R1

A	C
a	3
b	5

R2

R2.A is a foreign key references to R1.A
If on [update | delete] cascade has been set on R2.A, then update or delete to R1.A WILL cause the same change in R2.A

Set Null

A	B
a	1
b	2

R1

A	C
a	3
b	5

R2

R2.A is a foreign key references to R1.A
If on [update | delete] set null has been set on R2.A, then update or delete to R1.A WILL set the corresponding values in R2.A to null.