SQL – Displaying Query Results

Dr. Villanes

Questions
Q1—Q9

A few things...

- Errors in SQLite…
- Comments in SQLite
- Saving the queries in SQLite
- Slides are posted in Moodle

Presenting Data

Select Statement: Clauses

```
SELECT column1, column2, ...
FROM table_name
WHERE sql-expression
GROUP BY column_name
HAVING sql-expression
ORDER BY column_name <DESC>;
```

- The **WHERE** clause specifies data that meets certain conditions.
- The **GROUP BY** clause groups data for processing.
- The **HAVING** clause specifies groups that meet certain conditions.
- The **ORDER BY** clause specifies an order for the data.

The specified order of the above clauses within the SELECT statement is required.

Ordering Rows

Use the ORDER BY clause to order the query results.

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1;
```

ORDER BY *order-by-item* <**DESC**>
<,…*order-by-item* <**DESC**>>

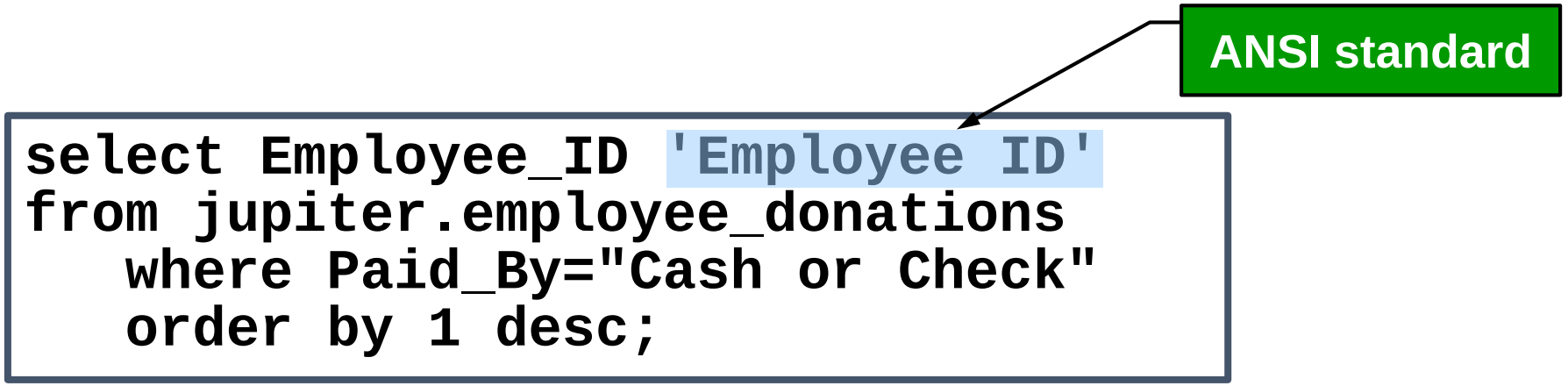
The default sort order when using an ORDER BY clause is ascending (no keyword or **ASC**). Use the **DESC** keyword following the column name to reverse the order.

Ordering Rows

- In an ORDER BY clause, ***order-by-item* is one of the following**:
  - a **column name** from any table in the FROM clause, even if the column is not in the SELECT clause
  - a **column alias**
  - an **integer** representing the position of an item in the SELECT clause
  - an **sql-expression**
- If more than one *order-by-item* is specified, then the first one determines the major sort order.

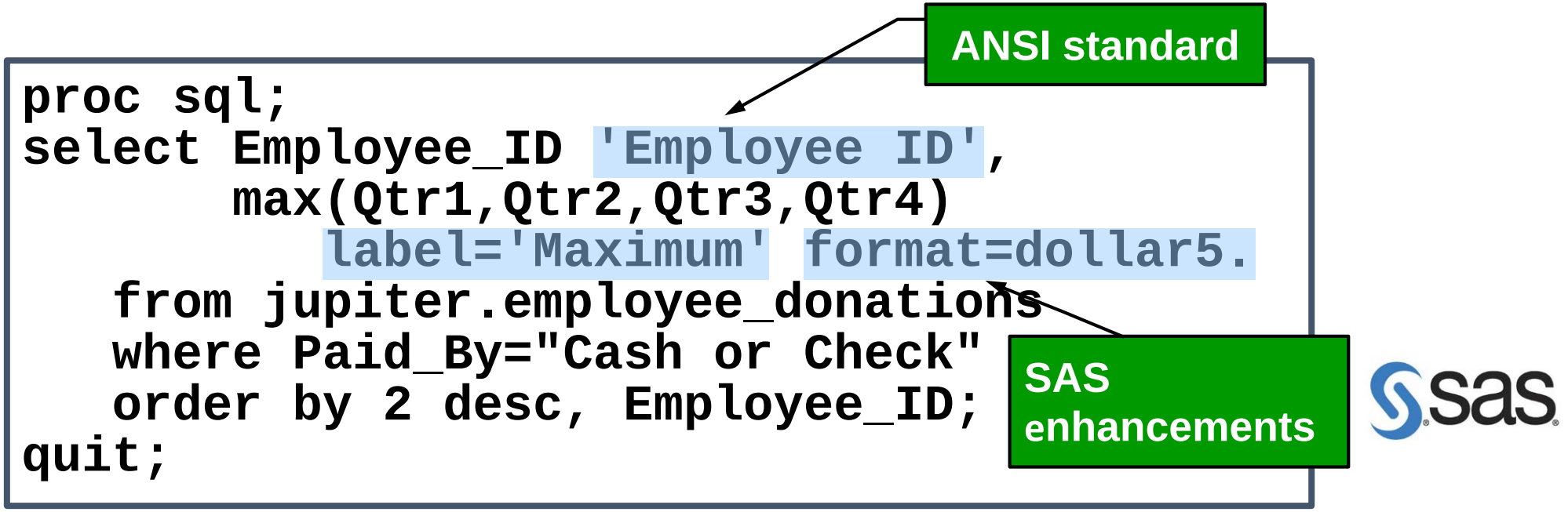
Adding Labels

- Column labels and formats must follow the column name and precede the comma.



Adding Labels/Format – SAS extension

- Column labels and formats must follow the column name and precede the comma.



Summarizing Data

Summary Functions: Down a Column

For a summary function with a single argument, nonmissing values are totaled down a column.

```
sum(Qtr1)
```

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4
120736	25	.	.	20
120759	15	20	5	.
120681	10	10	5	15
120679	.	20	5	15
120777	5	15	5	15

Commonly Used Summary Functions

ANSI SQL	SAS	Description
AVG	MEAN	Returns the mean (average) value.
COUNT	FREQ, N	Returns the number of nonmissing values.
MAX	MAX	Returns the largest value.
MIN	MIN	Returns the smallest nonmissing value.
SUM	SUM	Returns the sum of nonmissing values.
	NMISS	Counts the number of missing values.
	STD	Returns the standard deviation.
	VAR	Returns the variance.

Both ANSI SQL and SAS functions can be used in PROC SQL.

Summary Functions: COUNT Function

The *COUNT function* counts the number of rows returned by a query.

```
select count(*) as Count
   from employee_information;
```

COUNT(*argument*)

Argument value	Counts
* (asterisk)	All rows in a table or group
A column name	The number of nonmissing values in that column

Grouping Data

You can use the GROUP BY clause to do the following:

- classify the data into groups based on the values of one or more columns
- calculate statistics for each unique value of the grouping columns

```
select Employee_Gender as Gender,
       avg(Salary) as Average
from employee_information
group by Employee_Gender;
```

GROUP BY *group-by-item*<,…, *group-by-item*>

Grouping Data Example

- Classify the data into groups based on the values of one or more columns
- Calculate statistics for each unique value of the grouping columns

Employee

EmployeeID	Name	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

SELECT *DeptID, Avg(Salary)*
FROM *Employee*
GROUP BY *DeptID;*

GROUP BY

DeptID	Avg(Salary)
1	3000.0
2	4000.0
3	4250.0

Rule of thumb:

IF you list one or more than one column AND a summary function in the select clause…

THEN you need to specify a group by

What if I want to select which groups get displayed?

Employee

EmployeeID	Name	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

SELECT *DeptID, Avg(Salary)*
FROM *Employee*
GROUP BY *DeptID;*

GROUP BY

DeptID	Avg(Salary)
1	3000.0
2	4000.0
3	4250.0

Selecting Groups with the HAVING Clause

The ***HAVING clause*** **subsets groups based on the expression value.**

```
select Department, count(*) as Count
   from employee_information
   group by Department
   having Count ge 25
   order by Count desc;
```

GROUP BY *group-by-item <,…,group-by-item>*
HAVING *sql-expression*

Having Example

Employee

EmployeeID	Name	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

SELECT *DeptID, Avg(Salary) as Average*
FROM *Employee*
GROUP BY *DeptID*
HAVING *Average >= 4000;*

GROUP BY

DeptID	Avg(Salary)
2	4000.0
3	4250.0

WHERE Clause versus HAVING Clause

The WHERE clause is evaluated ***before*** a row is available for processing and determines which individual rows are available for grouping.

WHERE *sql-expression*

The HAVING clause is processed ***after*** the GROUP BY clause and determines which groups are displayed.

HAVING *sql-expression*