Practice (Continued)

- In the SASUSER library, create a new table called Employee_Organization2, which is an exact copy of the Employee_Organization table.
- Use the TRANWRD() function to change all occurrences of “Marketing Assistant ” to “MA_” of the Job_Title column of the Employee_Organization2 table.
- Use the SCAN() function, CONTAINS operator and concatenation operator to change all occurrences of “Sales Rep. ” to “SR_” of the Job_Title column of the Employee_Organization2 table, except those starting with “Temp.”

Code for the first bullet

Code for the second bullet

The third bullet

Use the SCAN() function, CONTAINS operator and CONCATENATION operator to change all occurrences of “Sales Rep. ” to “SR_” of the Job_Title column of the Employee_Organization2, except those starting with “Temp.”

Employee_ID	Job_Title
120121	SR_II
120122	SR_II
120123	SR_I

Third bullet: Connect all SR job titles to position levels with an underscore except those containing “Temp.”

Change “Temp. Sales Rep.” to “SR Temp.” using the TRANWRD() function

Find all the SR job titles that contain an underscore (_) using the LIKE keyword

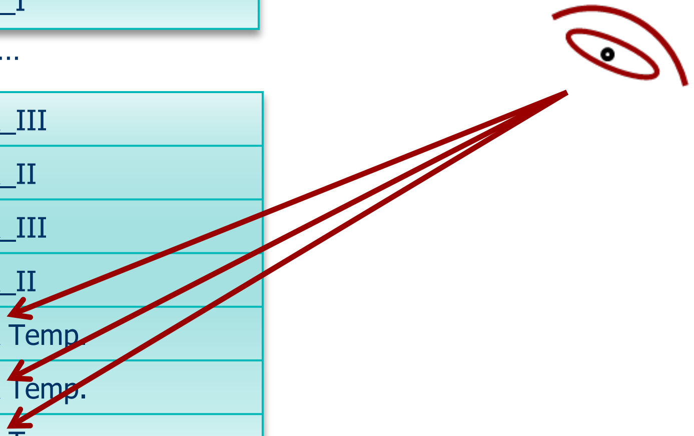
```
proc sql;
   select Employee_ID, Job_title
      from sasuser.Employee_Organization2
      where Job_title like 'SR_%';
quit;
```

Employee_ID	Job_Title
120121	SR_II
120122	SR_II
120123	SR_I

… …

120177	SR_III
120178	SR_II
120179	SR_III
120180	SR_II
120181	SR Temp.
120182	SR Temp.
120183	SR Temp.

… …



Why was that?

Because the WHERE clause uses the LIKE operator and the underscore represents any one character, not just an underscore (_) in the third position.

How to solve the problem?

The solution is to use an ESCAPE clause.

ESCAPE clause

To search for actual percent or underscore characters in your text using the LIKE operator, you must use an ESCAPE clause.

The ESCAPE clause in the LIKE condition enables you to designate a single character string literal, '/', known as an *escape character*, to indicate how PROC SQL should interpret the LIKE wildcards (% and _) when SAS is searching within a character string.

An example of the ESCAPE clause

```
proc sql;
   select Employee_ID, Job_title
      from sasuser.Employee_Organization2
      where Job_title like 'SR/_%' ESCAPE '/';
quit;
```

Employee_ID	Job_Title
120121	SR_II
120122	SR_II
120123	SR_I
120124	SR_I
120125	SR_IV
120126	SR_II
120127	SR_II
120128	SR_IV
120129	SR_III
120130	SR_I
120131	SR_I
120132	SR_III
120133	SR_II
120134	SR_II
120135	SR_IV
120136	SR_I
120137	SR_III
120138	SR_I
120139	SR_II
120140	SR_I
120141	SR_II
120142	SR_III
120143	SR_II
120144	SR_III
120145	SR_II
120146	SR_I
120147	SR_II
120148	SR_III
…	…

Multiple choice poll

Which of the following WHERE clauses correctly selects rows with a **`Job_Code`** value that begins with an underscore?

a. `where Job_Code like '%_%'`

b. `where Job_Code contains '%_%'`

c. `where Job_Code like '/_%' escape '/'`

d. `where Job_Code like '%/_%' escape '/'`

The CASE expression for conditional processing

General form of CASE expression:

```
CASE <case-operand>
    WHEN when-condition THEN result-expression
    <...WHEN when-condition THEN result-expression>
    <ELSE result-expression>
END;
```

Using different expressions to modify values for different subsets of rows within a column.

ELSE clause is strongly recommended.

where

CASE performs conditional processing.

case-operand is an optional expression that resolves to a table column whose values are compared to all the *when-conditions*.

WHEN specifies a *when-condition*, a shortened expression that assumes *case-operand* as one of its operands, and that resolves to true or false.

THEN specifies a *result-expression*, an expression that resolves to a value.

ELSE specifies a *result-expression*, which provides an alternate action if none of the *when-conditions* is executed.

END indicates the end of the CASE expression.

Two methods to update the salary column

Method of Updating Table	Example

use *multiple UPDATE statements*, one for each subset of rows
 	
A single UPDATE statement can contain only a single WHERE clause, so multiple UPDATE statements are needed to specify expressions for multiple subsets of rows.
	`proc sql;`
	`update work.payrollmaster_new`
	`set salary=salary*1.05`
	`where substr(jobcode,3,1)='1';`
	`update work.payrollmaster_new`
	`set salary=salary*1.10`
	`where substr(jobcode,3,1)='2';`
	`update work.payrollmaster_new`
	`set salary=salary*1.15`
	`where substr(jobcode,3,1)='3';`

use a *single UPDATE statement* that contains a *CASE expression*	`proc sql;`
	`update work.payrollmaster_new`
	`set salary=salary*`
	`case`
	`when substr(jobcode,3,1)='1'`
	`then 1.05`
	`when substr(jobcode,3,1)='2'`
	`then 1.10`
	`when substr(jobcode,3,1)='3'`
	`then 1.15`
	`else 1.08`
	`end;`

The first method uses multiple UPDATE statements and table must be read three times. The second method is more efficient and is recommended.

How PROC SQL Updates Rows Based on a CASE Expression

```
proc sql;
   update work.payrollmaster_new
      set salary=salary*
         case
              when substr(jobcode,3,1)='1'
                 then 1.05
              when substr(jobcode,3,1)='2'
                 then 1.10
              when substr(jobcode,3,1)='3'
                 then 1.15
              else 1.08
         end;
```

1. In the CASE expression, PROC SQL finds the WHEN-THEN clause that contains a condition that the row matches.

2. The CASE expression then returns the result from the matching WHEN-THEN clause to the SET clause. The returned value completes the expression in the SET clause.

3. The SET clause uses the completed expression to update the value of the specified column in the current row.

The WHEN-THEN clauses in the CASE expression are evaluated sequentially. When a matching case is found, the THEN expression is evaluated and set, and the remaining WHEN cases are *not* considered.

Using vs. without using CASE operand

Without operand

```
proc sql;
   update work.payrollmaster_new2
      set salary=salary*
         case
            when substr(jobcode,3,1)='1'
                   then 1.05
            when substr(jobcode,3,1)='2'
                   then 1.10
            when substr(jobcode,3,1)='3'
                   then 1.15
            else 1.08
         end;
```

With operand

```
proc sql;
   update work.payrollmaster_new2
      set salary=salary*
         case substr(jobcode,3,1)
            when '1'
                 then 1.05
            when '2'
                 then 1.10
            when '3'
                 then 1.15
            else 1.08
         end;
```

The SUBSTR function
is evaluated only once.

The case operand syntax might be used *only* if the WHEN clause expression uses the equals (=) comparison operator.

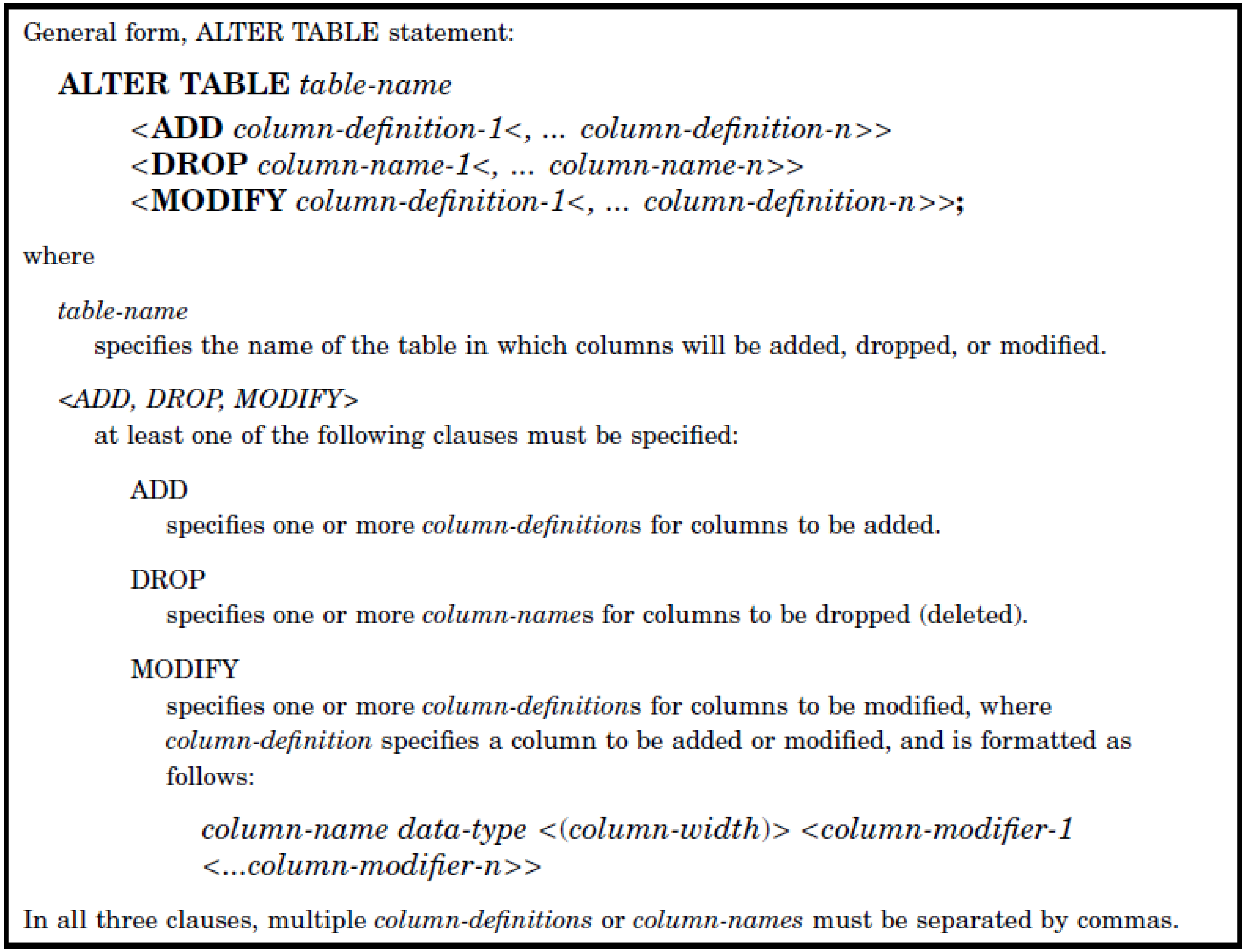
An example of using CASE expression in the SELECT clause

Assign the values of “junior,” “intermediate” and “senior” to **JobLevel**, based on the number at the end of each job code, which is 1, 2, or 3 respectively.

```
proc sql outobs=10;
  select lastname, firstname, jobcode,
     case substr(jobcode,3,1)
           when '1'
              then 'junior'
           when '2'
              then 'intermediate'
           when '3'
              then 'senior'
            else 'none'
      end as JobLevel
   from sasuser.payrollmaster, sasuser.staffmaster
   where staffmaster.empid= payrollmaster.empid;
```

LastName	FirstName	JobCode	JobLevel
ADAMS	GERALD	TA2	intermediate
ALEXANDER	SUSAN	ME2	intermediate
APPLE	TROY	ME1	junior
ARTHUR	BARBARA	FA3	senior
AVERY	JERRY	TA3	senior
BAREFOOT	JOSEPH	ME3	senior
BAUCOM	WALTER	SCP	none
BLAIR	JUSTIN	PT2	intermediate
BLALOCK	RALPH	TA2	intermediate
BOSTIC	MARIE	TA3	senior

Altering Columns in a Table



General form, ALTER TABLE statement:

> **ALTER TABLE** *table-name*
>
> <**ADD** *column-definition-1*<, ... *column-definition-n*>>
> <**DROP** *column-name-1*<, ... *column-name-n*>>
> <**MODIFY** *column-definition-1*<, ... *column-definition-n*>>**;**

where

table-name
specifies the name of the table in which columns will be added, dropped, or modified.

<ADD, DROP, MODIFY>
at least one of the following clauses must be specified:

ADD
specifies one or more *column-definitions* for columns to be added.

DROP
specifies one or more *column-names* for columns to be dropped (deleted).

MODIFY
specifies one or more *column-definitions* for columns to be modified, where *column-definition* specifies a column to be added or modified, and is formatted as follows:

> *column-name data-type* <(*column-width*)> <*column-modifier-1*
> <...*column-modifier-n*>>

In all three clauses, multiple *column-definitions* or *column-names* must be separated by commas.

Adding Columns

- Create a temporary table work.payrollmaster4, an exact copy of the sasuser.payrollmaster table.
- Add a column called Bonus with a data type of num, format = comma10.2
- Add a column called Level with data type of char(3)
- Display the new table

```
proc sql;
        alter table work.payrollmaster4
                add Bonus num format=comma10.2,
                    Level char(3);
        select * from payrollmaster4;
quit;
```

Adding Columns: the result

DateOfBirth	DateOfHire	EmpID	Gender	JobCode	Salary	Bonus	Level
16SEP1958	07JUN1985	1919	M	TA2	$48,126	.	
19OCT1962	12AUG1988	1653	F	ME2	$49,151	.	
08NOV1965	19OCT1988	1400	M	ME1	$41,677	.	
04SEP1963	01AUG1988	1350	F	FA3	$46,040	.	
16DEC1948	21NOV1983	1401	M	TA3	$54,351	.	
29APR1952	11JUN1978	1499	M	ME3	$60,235	.	
09JUN1960	04OCT1988	1101	M	SCP	$26,212	.	
03APR1959	14FEB1979	1333	M	PT2	$124,048	.	

…

Dropping Columns

- Drop the newly added columns, Bonus and Level from the work.payrollmaster4 table
- Check the content of the modified table

```
proc sql;
          alter table work.payrollmaster4
                    drop bonus, level;
           select * from payrollmaster4;
quit;
```

Dropping Columns: the result

DateOfBirth	DateOfHire	EmpID	Gender	JobCode	Salary
16SEP1958	07JUN1985	1919	M	TA2	$48,126
19OCT1962	12AUG1988	1653	F	ME2	$49,151
08NOV1965	19OCT1988	1400	M	ME1	$41,677
04SEP1963	01AUG1988	1350	F	FA3	$46,040
16DEC1948	21NOV1983	1401	M	TA3	$54,351
29APR1952	11JUN1978	1499	M	ME3	$60,235
09JUN1960	04OCT1988	1101	M	SCP	$26,212
03APR1959	14FEB1979	1333	M	PT2	$124,048

…

Modifying Columns in a Table

- You can modify a column's length (for character column), informat, format, and label.
- You cannot use the MODIFY clause to change a column's data type and name.
- You never need to specify data type for numerical columns and is optional for character columns; however, you have to specify data type when you change the width of a character column.

```
proc sql;
   alter table work.payrollmaster4
      modify jobcode char(2)
   select * from payrollmaster4;
quit;
```

```
proc sql;
  alter table work.payrollmaster4
      modify salary format=dollar11.2
             label="Salary Amt";
  select * from payrollmaster4;
quit;
```

When the new length is too small, the values may be truncated.

Practice: a business scenario

- Conditionally calculate a raise for employees based on the employee's job title.
  - Level I employees receive a 5% raise.
  - Level II employees receive a 7% raise.
  - Level III employees receive a 10% raise.
  - Level IV employees receive a 12% raise.
  - Everyone else receives an 8% raise.
- The Employee_payroll and Employee_organization tables contain all of the information that you need to calculate the salary changes.
- You are required to
  - create a table called employee_payroll2, containing three columns: Job_title, Salary and a column specified in the following bullets, in SASUSER by using a query.
  - use the CASE expression and the SCAN function to conduct the salary changes (do this in an SELECT clause).
  - create a new column called new_salary.
  - in a separate statement, change the format of the new_salary and salary columns to dollar12.2, and label the new_salary column as "Raised Salary" and the salary column as "Old Salary."
  - display the whole modified table.
  - delete the column labeled with "Old Salary" and change the label "Raised Salary" to "Salary."
  - display the whole modified table again.