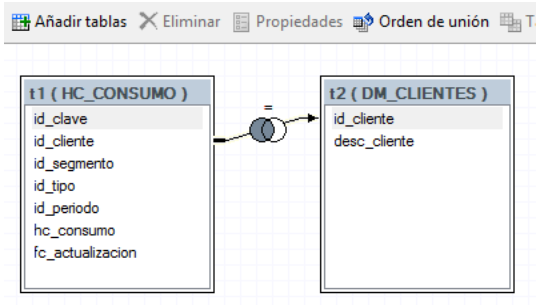


Nombre de columna	Columna de origen	Resumen
cod_periodo (cod_periodo)	t1.cod_periodo	
media_consumo	Calculado	AVG
suma_consumo	Calculado	SUM

Combinación y agregación de datos

sas

datademy

Unir conjuntos de datos

Unir conjuntos de datos mediante paso DATA:

```
data tabunion;
    set tabsas.consumo_1
        tabsas.consumo_2
        tabsas.consumo_3;
run;

proc append base=tabsas.consumo_1 new=tabsas.consumo_2;
run;
```

cod_periodo	cod_cliente	cod_tarifa	cod_segmento
201401	100058	100	S1
201401	100060	101	S2
201401	100060	101	S2
201401	100061	105	S5

cod_periodo	cod_cliente	cod_tarifa	cod_segmento
201401	100058	100	S1
201401	100060	101	S2
201401	100060	101	S2
201401	100061	105	S5

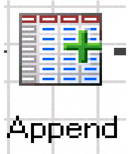
cod_periodo	cod_cliente	cod_tarifa	cod_segmento
201401	100058	100	S1
201401	100060	101	S2
201401	100060	101	S2
201401	100061	105	S5
201401	100058	100	S1
201401	100060	101	S2
201401	100060	101	S2
201401	100061	105	S5

- Empleando la sentencia **SET** podemos unir N tablas en una tabla destino:

```
DATA <tabla_destino>;
SET <tabla_origen1>
        <tabla_origen2>
         …..
         <tabla_origenN>;
run;
```

- Otra opción es emplear el procedimiento **APPEND,** con la diferencia de que este sólo permite unir dos tablas:

```
PROC APPEND base=<tabla_base>
                      new=<tabla_a_añadir>;
run;
```



Combinar conjuntos de datos / Unir conjuntos de datos

Unir conjuntos de datos mediante paso DATA :

• Con la sentencia **SET** podemos unir N tablas con diferente estructura de campos. Aquellos campos que no sean comunes a las tablas de entrada se quedarán con nulos en aquellos registros que procedan de las tablas de entrada que no contengan este campo. El proceso va a leer cada registro de las tablas de entrada.

• En el procedimiento **APPEND** añadimos registros sobre una base y el procedimiento espera que los nuevos registros se ajusten a la estructura de la tabla de base**.** A nivel de rendimiento es más rápido que el SET ya que no procesa cada registro de la entrada, simplemente añade nuevos registros al final

cod_periodo	cod_cliente	cod_tarifa	cod_segmento
201401	100058	100	S1
201401	100060	101	S2
201401	100060	101	S2
201401	100061	105	S5

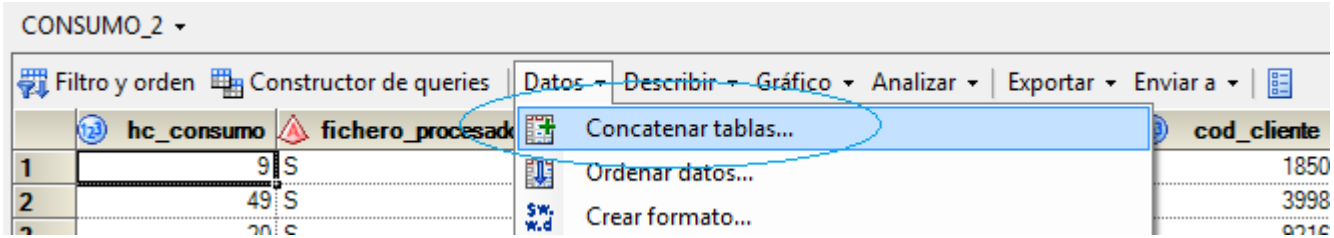
cod_periodo	cod_cliente	cod_tarifa	cod_segmento
201401	100058	100	S1
201401	100060	101	S2
201401	100060	101	S2
201401	100061	105	S5

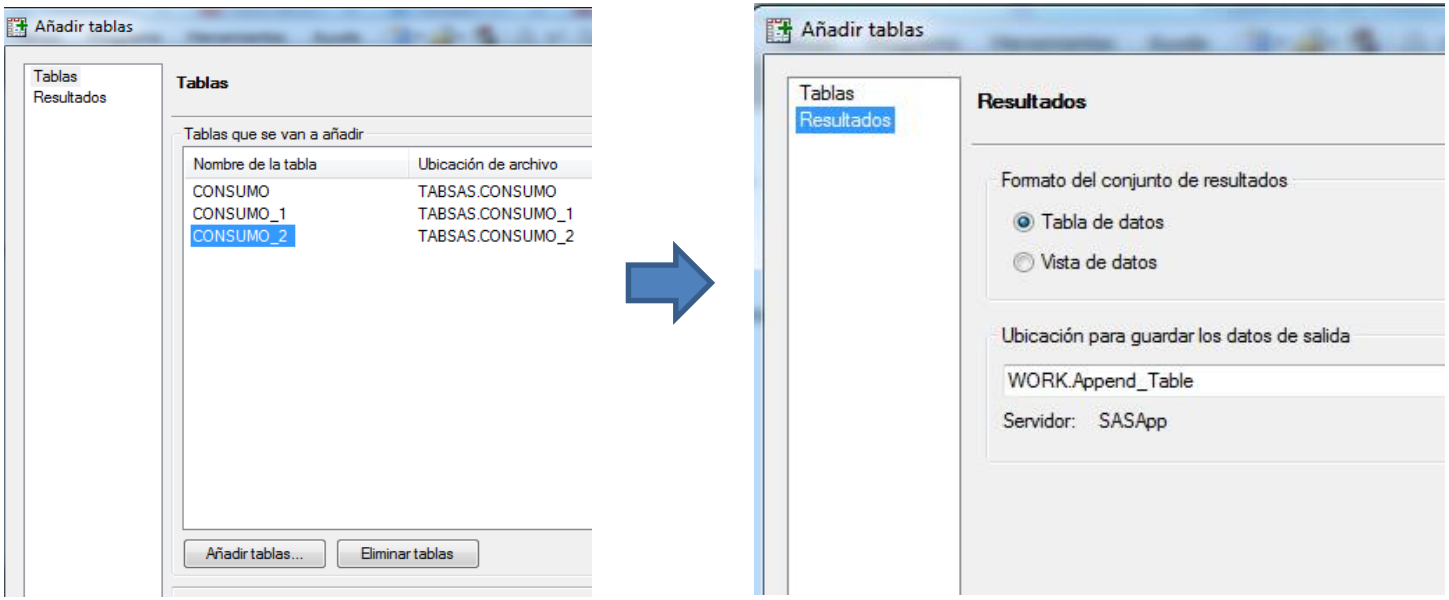
cod_periodo	cod_cliente	cod_tarifa	cod_segmento
201401	100058	100	S1
201401	100060	101	S2
201401	100060	101	S2
201401	100061	105	S5
201401	100058	100	S1
201401	100060	101	S2
201401	100060	101	S2
201401	100061	105	S5

Unir conjuntos de datos

Unir conjuntos de datos mediante tarea E. Guide:

- Para unir tablas utilizamos la tarea ‘Concatenar tablas’, en la que indicamos las tablas a unir y la tabla de resultados donde guardamos el resultado:





Combinar conjuntos de datos

Paso DATA combinación conjuntos de datos:

```
proc sort data=tabsas.tarifas;
by cod_tarifa;
run;

proc sort data=tabsas.consumo_5;
by cod_tarifa;
run;

data tabsas.consumo_6;
merge tabsas.tarifas tabsas.consumo_5;
by cod_tarifa;
run;
```

cod_periodo	cod_cliente	cod_tarifa	cod_segmento
201401	100058	100	S1
201401	100060	101	S2
201401	100060	101	S2
201401	100061	105	S5

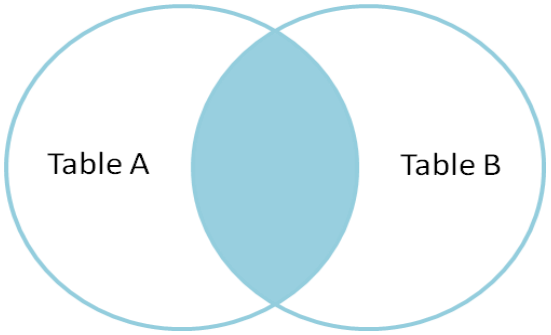
cod_periodo	cod_trimestre	cod_agno
201401	20141	2014
201401	20141	2014
201401	20141	2014
201401	20141	2014

cod_periodo	cod_cliente	cod_tarifa	cod_segmento	cod_trimestr	cod_agno
201401	100058	100	S1	20141	2014
201401	100060	101	S2	20141	2014
201401	100060	101	S2	20141	2014
201401	100061	105	S5	20141	2014

- Empleando la sentencia **MERGE** podemos combinar dos tablas en una tabla destino, indicando el campo de cruce:

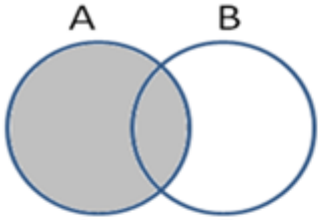
```
DATA <tabla_destino>;
MERGE <tabla_a>
          <tabla_b>;
BY <campo>;
run;
```

- Es necesario que las tablas estén previamente ordenadas por los campos de cruce:

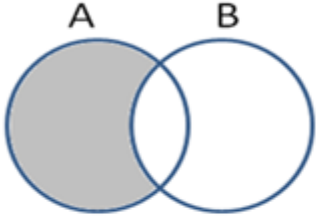


Combinar conjuntos de datos

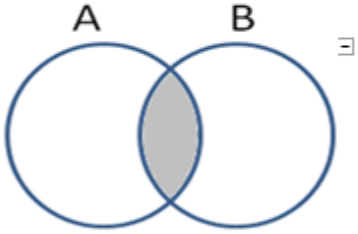
Tipos de cruces y salidas en un MERGE



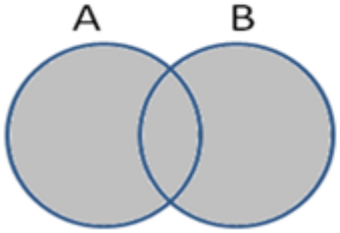

```
data tab_res;
 merge tab_a(in=a) tab_b(in=b);
 by campo;
 if a;
 run;
```



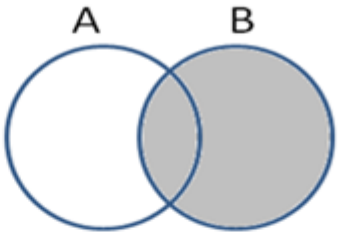

```
data tab_res;
 merge tab_a(in=a) tab_b(in=b);
 by campo;
 if a and not b;
 run;
```



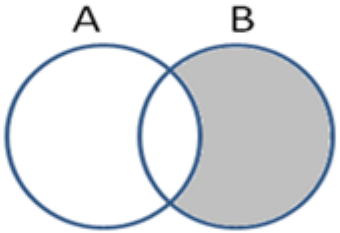

```
data tab_res;
 merge tab_a(in=a) tab_b(in=b);
 by campo;
 if a and b;
 run;
```



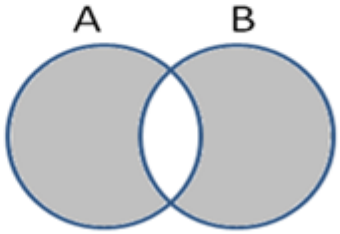

```
data tab_res;
 merge tab_a(in=a) tab_b(in=b);
 by campo;
 run;
```




```
data tab_res;
 merge tab_a(in=a) tab_b(in=b);
 by campo;
 if b;
 run;
```




```
data tab_res;
 merge tab_a(in=a) tab_b(in=b);
 by campo;
 if not a and b;
 run;
```




```
data tab_res;
 merge tab_a(in=a) tab_b(in=b);
 by campo;
 if not (a and b);
 run;
```

Combinar conjuntos de datos

Diferencias MERGE vs SQL JOIN:

MERGE	SQL JOIN
Las tablas deben estar previamente ordenadas	No necesaria ordenación previa
Permite incluir sentencias de control del tipo if/else, select/when para incluir lógica de negocio compleja	Permite incluir sentencias CASE, pero está más limitado a la hora de incluir lógica de negocio compleja.
Se pueden crear múltiples tablas de salida	Sólo permite una tabla de salida
Necesita que los campos de cruce se llamen igual	No es necesario que los campos de cruce se llamen igual
Algoritmo de cruce merge join	El algoritmo de cruce varía: por defecto hash join, si están ordenadas merge join, si hay índice cruce por índices

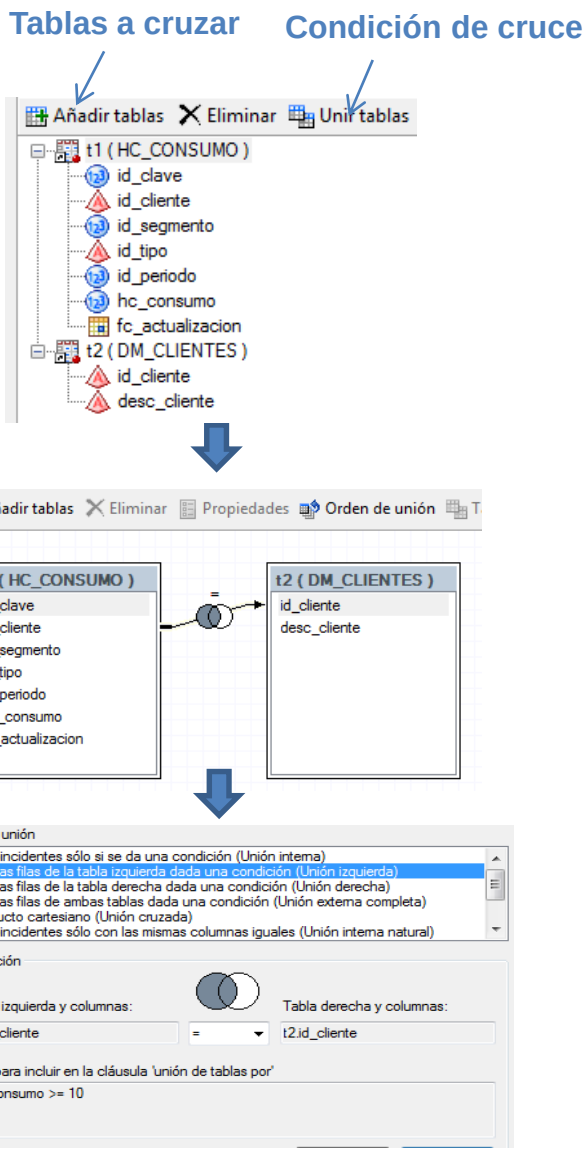
```
data tabsas.consumo_6;
merge tabsas.consumo_5(in=a) tabsas.tarifas(in=b);
by cod_tarifa;
if a and b;
run;
```

```
proc sql;
create table tabsas.consumo_7 as
(select a.*, b.* from tabsas.consumo_5 a, tabsas.tarifas b
where a.cod_tarifa = b.cod_tarifa);
quit;
```

Combinar conjuntos de datos

Combinar conjuntos de datos con constructor de queries:

- Para realizar **cruces de tablas** indicamos las tablas a cruzar, los campos de cruce, el tipo de cruce (inner join, full join, right join, left join….) y los campos seleccionados. Así mismo, podemos añadir condiciones sobre los registros de las tablas del cruce



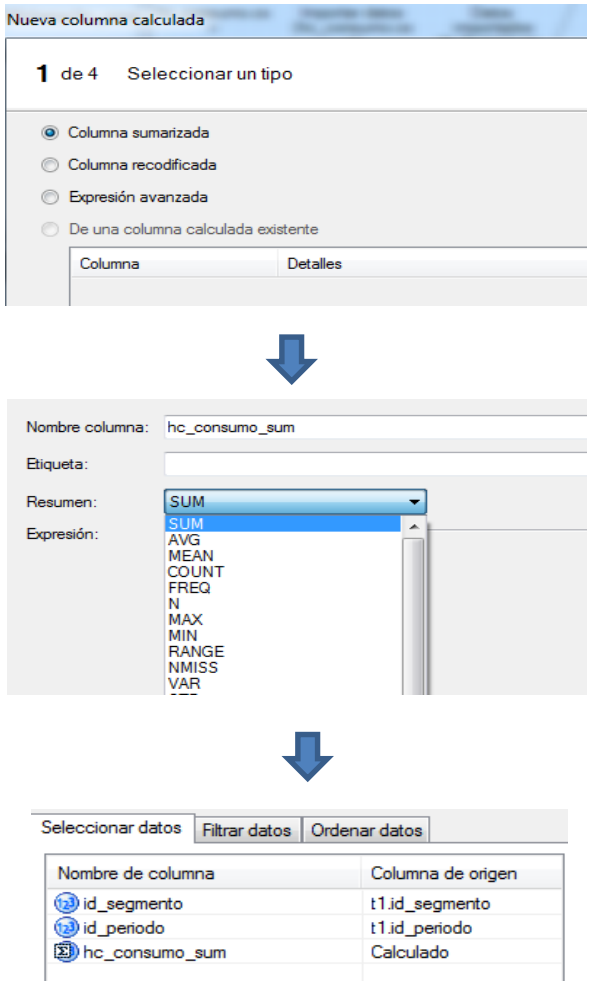

```
PROC SQL;
    CREATE TABLE WORK."HC_CONSUMO2"n AS
    SELECT t1.id_cliente,
           t2.desc_cliente,
           t1.id_segmento,
           t1.id_tipo,
           t1.id_periodo,
           t1.hc_consumo
      FROM WORK.HC_CONSUMO t1
           LEFT JOIN WORK.DM_CLIENTES t2 ON
(t1.id_cliente = t2.id_cliente AND (hc_consumo >= 10));
QUIT;
```

Agregación de conjuntos de datos

Construcción de queries:

- Para realizar **agregaciones** indicamos los campos sobre lo que realizar la agregación (columnas del group by) y el campo sobre el que realizamos el cálculo agregado, este campo se genera como columna calculada.
- Especificamos el cálculo a realizar sobre el conjunto de registros de cada clave: SUM, AVG, COUNT, FREQ, MIN, MAX……

```
PROC SQL;
    CREATE TABLE WORK.QUERY_FOR_HC_CONSUMO2 AS
    SELECT t1.id_segmento,
           t1.id_periodo,
           /* hc_consumo_sum */
             (SUM(t1.hc_consumo)) FORMAT=BEST2. AS hc_consumo_sum
      FROM WORK.HC_CONSUMO2 t1
      GROUP BY t1.id_segmento,
               t1.id_periodo;
QUIT;
```



Agregación de conjuntos de datos

Agregación conjuntos de datos:

```
 ; /* proc summary */
 proc summary data=tabsas.consumo_4 nway;
    class cod_periodo cod_tarifa;
    var hc_consumo;
    output out=ag_consumo1(drop=_type_ _freq_)
sum=hc_consumo ;
 run;

 /* proc means */
 proc means data=tabsas.consumo_4 noprint nway;
    class cod_periodo cod_tarifa;
    var hc_consumo;
    output out=ag_consumo2(drop=_type_ _freq_)
    sum(hc_consumo) = suma_consumo
    mean(hc_consumo) = media_consumo
    max(hc_consumo) = max_consumo ;
 run;
```

- Los procedimientos **MEANS** y **SUMMARY** permiten realizar agregaciones en base a los campos clave indicados. Igualmente permiten obtener otros valores estadísticos: máximo, mínimo, media, varianza, desviación estándar…

CLASS: campos clave de la agregación.

VAR: indicadores sobre los que obtener estadísticos.

OUTPUT: tabla de salida

Valores estadísticos: SUM, MAX, MEAN, STD, VAR…

TYPE: Variable que nos permite controlar los niveles de agregación

FREQ: Indica el número de registros de cada nivel de agregación