SQL - Lenguaje Estructurado de Consultas

en R y SAS

César Mignoni

Introducción

El lenguaje estructurado de consultas **SQL** (*Structured Query Language*) surge como un leguaje para realizar consultas en bases de datos relacional, modelo definido por el Dr. Edgar F. Codd de IBM en la década de los 70´, luego fue incorporado por muchos lenguajes de programación como una herramienta que permite realizar consultas y obtener conjuntos de datos de una manera rápida con una sintaxis sencilla. Actualmente, se lo puede utilizar en **SAS** mediante el procedimiento *Proc sql* y en **R** con varios paquetes, uno de ellos *sqldf*.

Es un lenguaje declarativo en el que las órdenes especifican cual debe ser el resultado y no la manera de conseguirlo, como ocurre en los lenguajes imperativos o procedimentales. Al ser declarativo, su aprendizaje es muy sencillo debido a que permiten escribir las órdenes, utilizando palabras clave (en inglés), como si fueran frases en las que se especifica que es lo que se quiere obtener. Por ejemplo:

```
SELECT nomemp FROM empleados WHERE sigladepto='prod' ORDER BY nomemp;
```

La sentencia anterior devuelve el nombre de aquellos empleados que pertenecen al departamento producción (*prod*) y los presenta ordenados por nombre.

Los comandos que contiene el *Lenguaje Estructurado de Consultas (SQL)* permiten realizar diversos tipos de tareas:

- Comandos para la **definición y creación** de un conjunto de datos (create table).
- Comandos para **inserción, borrado o modificación** de datos (insert, delete, update).
- Comandos para la **consulta** de datos, en algunos casos la selección es de acuerdo a criterios complejos que involucran diversas tablas relacionadas por un campo común (*select*).
- **Capacidades aritméticas**: En SQL es posible incluir operaciones aritméticas así como comparaciones, por ejemplo $A > B + 3$.
- **Asignación y comandos de impresión**: es posible imprimir una tabla construida por una consulta o almacenarla como una nueva tabla.
- **Funciones de agregación**: Operaciones tales como promedio (average), suma (sum), máximo (max), etc. se pueden aplicar a las columnas de una tabla para obtener una cantidad única y, a su vez, incluirla en consultas más complejas.

En una base de datos relacional, los resultados de la consulta van a ser datos individuales, tuplas o tablas, generados a partir de un comando para consultas en el que se establecen una serie de condiciones. Por ejemplo una típica consulta sobre una tabla en una base de datos relacional, utilizando *SQL* podría ser:

```
SELECT numemp, nomemp, cargoid
FROM empleados
WHERE sigladepto='prod';
```

El resultado será un listado que tendrá tres columnas (numemp, nomemp, cargoid) provenientes de la tabla empleados, las filas corresponderán sólo a aquellos casos en los que el departamento (columna sigladepto) sea producción (='prod'). En el caso de que sólo uno de los empleados cumpliera la condición se obtendría una sola fila como salida.

Componentes del SQL

El lenguaje *SQL* está compuesto por comandos, cláusulas, operadores y funciones de agregación. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

Comandos

Existen dos tipos de comandos SQL:

- Los que permiten crear y definir nuevas tablas, campos e índices en las bases de datos.

CREATE	Utilizado para crear nuevas tablas y campos.
DROP	Empleado para eliminar tablas.
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

- Los que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.
INSERT	Utilizado para cargar datos en la base de datos en una única operación.
UPDATE	Utilizado para modificar los valores de los campos y registros especificados.
DELETE	Utilizado para eliminar registros de una tabla de una base de datos.

Cláusulas

Las cláusulas son condiciones utilizadas para indicar cuales son los datos que se quieren seleccionar o manipular.

FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros.
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.
GROUP BY	Utilizada para clasificar los registros seleccionados en grupos específicos.
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo.
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico.

Operadores

Operadores Lógicos

AND	Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT	Devuelve el valor contrario de la expresión.

Operadores de Comparación

<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor o Igual que
>=	Mayor o Igual que
=	Igual que

BETWEEN	Permite especificar un intervalo de valores.
LIKE	Compara una cadena de texto con una expresión regular.

Funciones de Agregación

Las funciones de agregación se usan dentro de una cláusula SELECT para devolver un único valor que se aplica a un grupo de registros.

AVG	Calcula el promedio de los valores de un campo determinado.
COUNT	Devuelve el número de registros de la selección.
SUM	Suma los valores de un campo determinado.
MAX	Devuelve el valor más alto de un campo especificado.
MIN	Devuelve el valor más bajo de un campo especificado.

Manipulación de datos en SAS utilzando SQL

El conocimiento obtenido hasta ahora sobre SQL es suficiente para comenzar a usarlo en SAS.

Como se mencionó en la intro de este material el procedimiento *PROC SQL* es la sentencia **SAS** que permite ejecutar las estructuras del lenguaje **SQL** encerrando las cláusulas del mismo.

```
proc sql;
select pais,
       poblac,
       relig
    from Datos.Mundo;
quit;
```

Lo primero que hay que tener en cuenta sobre la sintaxis de *PROC SQL* es que toda la consulta (*SELECT*... *FROM*...) se trata como una sola declaración. Solo hay un punto y coma (;) colocado al final de la consulta. Esto es así sin importar la complejidad de la consulta o cuántas cláusulas contenga.

En segundo lugar, el procedimiento finaliza con una instrucción *QUIT* en lugar de una instrucción *RUN*. Las consultas se ejecutan de forma inmediata, tan pronto como cuando llega al punto y coma de la instrucción *SELECT*. Hay dos cosas a tener en cuenta: 1) una sola instancia de *PROC SQL* puede contener más de una consulta y 2) la instrucción *QUIT* no es necesaria para que se ejecuten las consultas.

Finalmente, las declaraciones *SQL* solo pueden ejecutarse dentro de *PROC SQL*. No se pueden colocar en otros procedimientos o en el código de un paso *Data*.

Los resultados que produce la consulta anterior se observa en la siguiente imagen que muestra una ventana output de SAS

PAIS	POBLAC	RELIG
Acerbaján	7400	Musulmana
Afganistán	20500	Musulmana
Alemania	81200	Protestante
Arabia Saudí	18000	Musulmana
Argentina	33900	Católica
Armenia	3700	Ortodoxa
Australia	17800	Protestante
Austria	8000	Católica
Bahrein	600	Musulmana
Bangladesh	125000	Musulmana
Barbados	256	Protestante
Bélgica	10100	Católica
Bielorusia	10300	Ortodoxa
Bolivia	7900	Católica
.	.	.
.	.	.
Ucrania	51800	Ortodoxa
Uganda	19800	Católica
Uruguay	3200	Católica
Uzbekistán	22600	Musulmana
Venezuela	20600	Católica
Vietnam	73100	Budista
Zambia	9100	Protestante

Las columnas se presentan en el orden en que se especificaron en SELECT y, de forma predeterminada, se utiliza la etiqueta de la columna (si existe).

Una salida distinta para los resultados

PROC SQL funciona igual que todos los demás procedimientos *SAS*, los resultados de una consulta se muestran de forma predeterminada en la ventana de output, pero es posible defnir varias salidas *ODS* para que los resultados de una consulta *SELECT* se muestren en todos los destinos *ODS* abiertos; el siguiente código muestra como sería:

```
ods html body="c:\temp\Mundo.html";
ods pdf file="c:\temp\Mundo.pdf";
proc sql;
  select pais,
         poblac,
         relig
      from Datos.Mundo;
quit;
ods html close;
ods pdf close;
```

Producirá resultados en los tres destinos *ODS* abiertos: la ventana de resultados (el destino *OUTPUT* está abierto de forma predeterminada), un archivo *HTML* y un archivo *PDF*.

También se puede crear un conjunto de datos SAS (una tabla) a partir de los resultados de la consulta precediendo la instrucción *SELECT* con una instrucción *CREATE TABLE*.

```
proc sql;
  create table MundoInfo as
  select pais,
         poblac,
```

```
            relig
        from Datos.Mundo;
quit;
```

La instrucción CREATE TABLE hace dos cosas:

1. Crea un nuevo conjunto de datos SAS.
2. Suprime la salida impresa de la consulta.

En cuanto a los destinos *ODS* que pudieran estar abiertos no se les daría importancia y no se generaría las salidas. El orden de las columnas en la instrucción *SELECT* no solo determina el orden en la salida de la consulta, sino que también determina el orden en una tabla si se usa *CREATE TABLE*.

Opciones de la declaración SELECT

Las columnas en una instrucción SELECT se pueden renombrar, etiquetar o reformatear.

- Cambiar nombre: usar 'AS *nuevo nombre*'
- Etiqueta: usar 'LABEL = *etiqueta deseada*'
- Formato: usar 'FORMAT = *formato deseado*
- Longitud: usar 'LENGTH = *amplitud deseada*

La siguiente consulta crearía un conjunto de datos SAS temporal llamado *MundoInfo* que tiene tres variables y cuyos nombres serán: *PS*, *PB* y *RELIG*

```
proc sql;
    create table MundoInfo as
    select pais as PS,
         poblac as PB,
         relig label='Religión País' length= 12
    from datos.mundo;
quit;
```

Cuando se crea una nueva tabla, todos los formatos y etiquetas de la tabla original se transfieren de forma predeterminada. En la consulta anterior se asocia un nuevo ancho de campo con *relig* y se le adjunta una nueva etiqueta como: *Religión País*. Aquí también se cambia el nombre a *pais* y *poblac*. Cualquier otro formato o etiqueta se mantendrá como estaba hasta el momento en la tabla de origen.

No hay restricción en el número de atributos que se pueden establecer en una columna. Se puede renombrar, reformatear y volver a etiquetar, todo al mismo tiempo. Simplemente hay que separar las asignaciones de atributos con espacios. En la consulta anterior a la variable *relig* se le cambia la etiqueta y el ancho.

Selección de todas las columnas

Para especificar en una consulta que se quieren obtener todas las columnas de una tabla se utiliza un asterisco (*) en lugar de una lista de las columnas. Por ejemplo:

```
proc sql;
    create table MundoCopia as
```

```
    select *
    from Datos.Mundo;
quit;
```

Esta forma rápida permite ahorrar tiempo, pero, se debe tener conocimiento previo de los datos para saber que es lo que se obtendrá. Se debe considerar que esta forma de acceso rápido puede ser problemático cuando se está trabajando con combinaciones de varias tablas y con las operaciones de conjuntos.

Creación de nuevas columnas

De la misma manera que se utilizó la palabra clave *AS* para cambiar el nombre de una columna, también se puede usar para nombrar una nueva columna. En el siguiente ejemplo se utilizan dos columnas existentes para crear una nueva columna y se observa que la sintaxis de la asignación es opuesta a la de una expresión aritmética normal usando el "*signo igual*", en este caso la expresión está en el lado izquierdo de "*AS*".

```
proc sql;
 Select sexo,
        edad,
        altura,
        peso,
       (peso/(altura/100)**2) as IMC
    from Datos.curso;
quit;
```

El resultado de la consulta anterior se vería de la siguiente manera:

Sexo alumno	Edad Alumno	Estatura alumno	Peso alumno	IMC
FEME	20	172	59	19.94321
FEME	21	180	66	20.37037
FEME	22	180	62	19.1358
FEME	22	176	80	25.82645
FEME	23	176	74	23.88946
FEME	20	175	77	25.14286
FEME	22	186	73	21.10071
FEME	21	183	74	22.09681
FEME	21	175	77	25.14286
FEME	21	180	90	27.77778
FEME	19	180	60	18.51852
MASC	20	163	60	22.58271
MASC	20	164	55	20.44914
FEME	20	180	64	19.75309
MASC	21	165	61	22.40588
FEME	24	172	89	30.08383
FEME	22	174	80	26.42357
FEME	25	170	67	23.18339
MASC	21	160	64	25
MASC	20	158	54	21.63115
MASC	21	163	56	21.0772
FEME	25	174	77	25.43269
FEME	20	178	73	23.04002
MASC	19	167	53	19.00391
FEME	20	174	65	21.46915

Se puede ver que la nueva columna en la consulta tiene la etiqueta "IMC", si se hubiera incluido una instrucción *CREATE TABLE* este habría sido el nombre de la variable. Si no se especifica *AS*, en la salida, la columna no habría tenido etiqueta, solo espacio en blanco, y el nombre de la variable en el conjunto de datos creado sería _TEMA001. Si para la creacion de nuevas columnas no se especifica "*AS nombre*", las mismas se llamarían _TEMA002, _TEMA003, etc. Observar la importancia de nombrar las nuevas columnas, principalmente cuando se esta creando un nuevo conjunto de datos o se esta haciendo una salida impresa.

Selección de filas con la cláusula WHERE

Una vez que se han definido las columnas que se quieren seleccionar para la consulta, es posible que no se quieran todas las filas del conjunto de datos de origen.

La cláusula WHERE brinda la manera de seleccionar filas, pues contiene la lógica condicional necesaria para determinar que fila se incluirá en la salida de la consulta. Puede contener cualquier expresión válida de SAS.

Dentro de la instrucción *SELECT FROM* la cláusula *WHERE* es opcional, si se incluye en una consulta debe ir siempre seguido a la cláusula FROM. El siguiente código SAS utiliza la clausula *WHERE* para seleccionar desde un conjunto de datos llamado "mundo" solo las filas que corresponden a los países que practican la religión "musulmana".

```
proc sql;
  select *
    from Datos.Mundo
    where relig eq 'Musulmana';
quit;
```

La sintaxis de la cláusula WHERE admite más de una condición de selección para hacer que un registro sea incluido en la salida de la consulta. Cuando debe cumplirse más de una condición se utilizan los operadores lógicos *AND* y *OR* para agruparlas. Por ejemplo, si se quieren restringir la selección de filas desde el conjunto de datos "mundo" seleccionando solo los países que practican la religión "musulmana" y con un índice de alfabetización mayor a 50, se debe agregar la segunda condición a la cláusula *WHERE*:

```
proc sql;
  select *
    from Datos.Mundo
      where relig eq 'Musulmana' and
        alfabet > 50;
quit;
```

> Nota: No hay límite para la cantidad de condiciones que puede tener la cláusula WHERE.

Existe una diferencia de cómo *SAS* maneja los tipos de datos cuando no coincidentes en las cláusulas *WHERE* y otras partes del lenguaje. Esto se puede ver cuando se utiliza *WHERE* en *PROC SQL* o en un paso *DATA* u otro procedimiento. En la mayoría de los casos SAS realiza una conversión del tipo de dato en forma automática, de carácter a numérico o viceversa, para que la comparación tenga válidez y no presente un error. En siguiente ejemplo de código *SAS*, suponiendo que la variable "TPO" (Tiempo demora) es de tipo numérica, el paso *DATA* se ejecutará presentando una nota aclaratoria, pero la consulta SQL no:

```
data ResulData;
  set Datos.curso;
  if tpo > '20';
run;
```

```
proc sql;
  create table ResulConsul as
  select *
    from Datos.curso
```

```
        where tpo > '20';
quit;
```

En el paso *DATA* la instrucción *IF* realiza una conversión automática de '20' a numérico y evalúa la expresión, pero escribe una *nota* en la ventana *Log* que indica lo siguiente:

```
NOTE: Character values have been converted to numeric values at the places given by:
    (Line) :(Column).
```

En una consulta SQL la cláusula *WHERE* requiere que el tipo de datos sea compatible, por tal motivo genera un error y no se ejecuta:

```
ERROR: Expression using greater than (>) has components that are of different data typ
```

El requisito de compatibilidad de datos de la cláusula *WHERE* es válido si se usa en una consulta *SQL*, paso *DATA* u otro procedimiento. Los mensajes de error escritos en la ventana Log pueden ser diferentes. Por ejemplo, al ejecutar el paso *DATA* anterior con una instrucción *WHERE* en lugar de una instrucción *IF* se genera un mensaje de error ligeramente diferente y el paso *DATA* no se ejecuta:

```
data ResulData;
  set Datos.curso;
  where tpo > '20';
run;

ERROR: WHERE clause operator requires compatible variables.
```

Operadores especiales de la cláusula WHERE

Hay una serie de operadores que solo se pueden usar en una cláusula *WHERE*. Algunos de ellos pueden simplificar y hacer más eficiente la programación porque reducen el código necesaio para realizar algunas operaciones.

- Los operadores *IS NULL* y *IS MISSING*

Es posible usar los operadores *IS NULL* o *IS MISSING* para que devuelvan las filas con valores faltantes. La ventaja de *IS NULL* (o *IS MISSING*) es que la sintaxis es la misma si la variable es caracter o numérica.

```
proc sql;
  select *
  from Datos.curso
  where tipotran is null;
quit;
```

> Nota: IS MISSING es una extensión específica de SAS para SQL.

En la mayoría de las implementaciones de bases de datos hay una distinción entre valores vacíos (faltantes) y valores nulos. Los valores nulos son un caso único y se comparan con éxito con cualquier cosa que no sean otros valores nulos. Se debe tener en cuenta cómo se manejan los valores nulos si se está utilizando SQL en un entorno que no sea SAS.

- El operador BETWEEN

El operador *BETWEEN* le permite buscar un valor que se encuentre entre otros dos valores.

```
proc sql;
  select *
  from Datos.curso
  where tpo between 20 and 40;
quit;
```

Cuando se usa *BETWEEN* hay que tener en cuenta que los puntos finales se incluyen en los resultados de la consulta. En el ejemplo anterior, 20 y 40 están incluidos. La variable utilizada con *BETWEEN* puede ser numérica o de caracteres. Hay que considerar los riesgos que existen al usar variables de tipo caracter en las comparaciones, es recomendable conocer cual es la clasificación que utiliza el sistema operativo, es decir, ¿"a" es mayor o menor que "A"? y también que, " a" no es lo mismo que "a".

Hay un comportamiento interesante del operador *BETWEEN*. Los valores se tratan como los límites de un rango y se colocan automáticamente en el orden correcto. Esto significa que las siguientes dos condiciones producen el mismo resultado:

```
where tpo between 20 and 40;
where tpo between 40 and 20;
```

El orden en que se colocan los valores entre no es importante, aunque se recomienda que se especifiquen en la secuencia correcta para facilitar la comprensión.

Ordenar los datos seleccionados

Lo visto hasta ahora de *SQL* podría hacerse en un paso *DATA*. Tanto *SELECT... FROM...*, como **WHERE**, juegan un papel importante para determinar qué información se incluirá en los resultados de la consulta. La cláusula **ORDER BY** toma las filas seleccionadas y las ordena. La cláusula **ORDER BY** es opcional y, si se usa, debe estar seguida a la cláusula **WHERE**; si **WHERE** no se usa, debe seguir a la cláusula **FROM**.

```
proc sql;
    select *
    from Datos.Mundo
    where relig eq 'Musulmana'
    order by pais;
quit;
```

Esta consulta seleccionaría todos los países que practican la religión Musulmana y los ordena por nombre del país.

Como se puede observar, la sintaxis de la cláusula **ORDER BY** es similar a la declaración **BY** de **PROC SORT**. Si se quieren ordenar las filas por más de una variable, las mismas se deben colocar en la secuencia de orden separadas por comas.

Opción DESC en ORDER BY

El orden de clasificación predeterminado es ascendente. Para obtener una clasificación descendente, se usa la opción **DESC** seguido al nombre de la variable.

```
proc sql;
    create table Mundo_ord as
```

```
    select relig,
         pais,
         alfabet
    from Datos.Mundo
    where alfabet < 50
    order by relig desc,
             pais;
quit;
```

La opción **DESC** se puede agregar a tantas variables como sea necesario, siempre después del nombre de la variable y antes de la coma o punto y coma.

Funciones agregadas

Se puede usar funciones de agregado (o resumen) para resumir los datos en las tablas. Estas funciones pueden actuar en múltiples columnas en una fila o en una sola columna a través de filas. La siguiente lista es de las funciones agregadas.

Avg promedio de valores
Count el número de valores que no faltan
CSS suma corregida de cuadrados
CV Coeficiente de variación
Freq (igual que Count)
Max Valor máximo
Mean (igual que Avg)
Min Valor mínimo
N (igual que Count)
NMiss número de valores faltantes
Prt probabilidad de un mayor valor absoluto de t de Stude
Range rango de valores
Std Desviación estándar
StdErr Error estándar de la media
Sum Suma de valores
T El valor t de Student para comprobar que la media de la población es 0
USS suma de cuadrados sin corregir
Var Variancia

Se ha visto el uso de funciones SAS en una instrucción SELECT para crear nuevas columnas basadas en el valor de una columna existente. Las *funciones agregadas* también crean nuevas columnas, ya sea resumiendo columnas en una sola fila o resumiendo una columna en varias filas.

Si se incluye más de una columna en la lista de argumentos de la función, esa operación se realiza para cada fila de la tabla. En la siguiente tabla, cada fila de la tabla localidades obtiene una nueva columna, TPob, que es la suma de Tvar y Tmuj.

```
proc sql;
  select localidad length=20,
       depart length=20,
       Tmuj,
       Tvar,
  sum(Tvar,Tmuj) as TPob
  from datos.localidades;
quit;
```

Si la lista de argumentos contiene una sola columna, la operación se realiza en todas las filas de la tabla. La siguiente consulta resumirá TPob, de diferentes maneras, para toda la tabla. Se observa que los resultados de esta consulta es una sola fila que contiene la suma, la media y los valores máximos de Población total por localidad (TPob), así como el número de filas con un valor faltante para la columna TPob, si hubiese.

```
proc sql;
  select sum(TPob) as TotalTPob,
         mean(TPob) as MeanTPob,
         max(TPob) as MaxTPob,
         nmiss(TPob) as TPobMiss
  from datos.Tpob_localidades;
quit;
```

Nota: se puede hacer referencia a tantas funciones agregadas como sea necesario en una sola consulta y no todas tienen que actuar en la misma columna.

En la consulta anterior se observa que no se enumeraron columnas aparte de las que usan las funciones de agregado. Recordar que se le está diciendo a SQL que resuma toda la columna de la tabla. ¿Cómo se interpretaría si también pidiéramos algunas otras columnas? Por ejemplo,

```
proc sql;
  select depart length=15,
        prov,
        sum(TPob) as TotalTPob length=7
  from datos.Tpob_localidades;
quit;
```

La consulta calcula la suma total de la población para toda la tabla y no discrimina al departamento ni a la provincia para el cálculo.

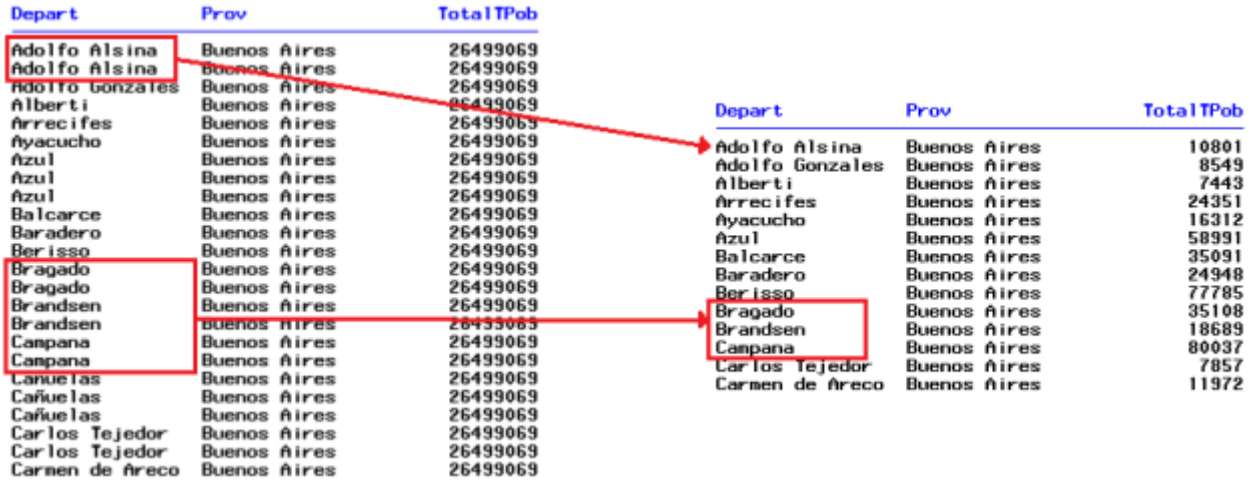
También recibirá una nota en la Log indicando que la consulta tuvo que ejecutarse más de una vez:

```
NOTE: The query requires remerging summary statistics back with the original data.
La cláusula GROUP BY
```

Por defecto, las funciones de resumen funcionan en todas las filas de una tabla. Se puede resumir por grupos de datos con la cláusula GROUP BY. Por ejemplo:

```
proc sql;
  select depart length=15,
       prov,
       sum(TPob) as TotalTPob length=7
  from datos.Tpob_localidades
  group by prov, depart;
quit;
```

Los resultados parciales de la consulta anterior se muestran aquí:



Depart	Prov	TotalTPob
Adolfo Alsina	Buenos Aires	26499069
Adolfo Alsina	Buenos Aires	26499069
Adolfo Gonzales	Buenos Aires	26499069
Alberti	Buenos Aires	26499069
Arrecifes	Buenos Aires	26499069
Ayacucho	Buenos Aires	26499069
Azul	Buenos Aires	26499069
Azul	Buenos Aires	26499069
Azul	Buenos Aires	26499069
Balcarce	Buenos Aires	26499069
Baradero	Buenos Aires	26499069
Berisso	Buenos Aires	26499069
Bragado	Buenos Aires	26499069
Bragado	Buenos Aires	26499069
Brandsen	Buenos Aires	26499069
Brandsen	Buenos Aires	26499069
Campana	Buenos Aires	26499069
Campana	Buenos Aires	26499069
Cañuelas	Buenos Aires	26499069
Cañuelas	Buenos Aires	26499069
Cañuelas	Buenos Aires	26499069
Carlos Tejedor	Buenos Aires	26499069
Carlos Tejedor	Buenos Aires	26499069
Carmen de Areco	Buenos Aires	26499069

Depart	Prov	TotalTPob
Adolfo Alsina	Buenos Aires	10801
Adolfo Gonzales	Buenos Aires	8549
Alberti	Buenos Aires	7443
Arrecifes	Buenos Aires	24351
Ayacucho	Buenos Aires	16312
Azul	Buenos Aires	58991
Balcarce	Buenos Aires	35091
Baradero	Buenos Aires	24948
Berisso	Buenos Aires	77785
Bragado	Buenos Aires	35108
Brandsen	Buenos Aires	18689
Campana	Buenos Aires	80037
Carlos Tejedor	Buenos Aires	7857
Carmen de Areco	Buenos Aires	11972

Se puede Observar que hay una fila por departamento y que la población total se ha resumido. Recordar de la sección anterior que sin la cláusula GROUP BY, la población total de la tabla se habría agregado a cada fila de la tabla.

El operador DISTINCT solo permite obtener una lista de los valores distintos para una o más columnas (variable) de una tabla y puede considerarse que tiene algunas de las propiedades de una función de resumen.

```
proc sql;
  select distinct relig
  from datos.mundo;
quit;
```

La consulta anterior devuelve una lista de todos los valores únicos de religión. No realiza un conteo, solo muestra una lista. Se pueden colocar varias columnas después del operador DISTINCT y se obtendrá una lista de todas las combinaciones de las columnas. Por ejemplo, la siguiente consulta devuelve los valores de las combinaciones únicas de marca y origen.

```
Proc sql;
  select distinct Make,
  origin
  from SASHELP.CARS;
quit;
```

Nota: Cuando se enumeran varias columnas con DISTINCT, solo se devuelven las combinaciones de valores reales, no todas las combinaciones posibles.

La siguiente consulta muestra el uso de DISTINCT dentro de la función COUNT. Esta consulta devolverá el número de valores únicos de religión, en este caso, 11.

```
proc sql;
  select count(distinct relig)
  from datos.mundo;
quit;
```

Nota: no se puede enumerar más de una columna en la función COUNT.

La cláusula HAVING

Para seleccionar filas en función de los resultados de una función de resumen se usa la cláusula HAVING. La cláusula HAVING es similar a la cláusula WHERE en que permite seleccionar filas para mantener en el conjunto de resultados de su consulta. Es una cláusula opcional y se coloca después de la cláusula GROUP BY.

```
proc sql;
  select depart length=15,
       prov,
       sum(TPob) as TotalTPob length=7
  from datos.Tpob_localidades
  group by prov, depart;
  having TotalTPob gt 50000;
quit;
```

En la consulta anterior, la condición HAVING (TotalTPob > 50000) se evalúa después de que la consulta se haya ejecutado y las filas se hayan agrupado y solo las filas que cumplan la condición se escribirán en el conjunto de resultados.

Es importante destacar la diferencia entre HAVING y WHERE. La cláusula WHERE selecciona filas a medida que entran en la consulta. Tiene que hacer referencia a columnas que existen en la tabla de consulta o que se calculan utilizando columnas en las tablas de consulta. No puede hacer referencia a columnas de resumen. HAVING, por otro lado, hace referencia a columnas de resumen a medida que las filas salen de la consulta.

¿Qué sucede si usa HAVING en columnas sin resumen? Depende mucho del resto de SELECT. Si no tiene columnas de resumen, las cláusulas WHERE y HAVING producirán resultados idénticos. Por ejemplo, estas dos consultas producen exactamente el mismo resultado.

```
proc sql;
    select pais,
    poblac,
    densidad
    from Datos.Mundo
    where relig eq 'Musulmana';
quit;
```

```
proc sql;
    select pais,
    poblac,
    densidad
    from Datos.Mundo
    having relig eq 'Musulmana';
quit;
```

No hay un resumen de las filas, por lo que los valores que entran y los que salen de la consulta son los mismos. Sin embargo, existe una gran diferencia cuando hay una función resumen en la instrucción SELECT y la cláusula HAVING hace referencia a una columna no resumida.

Manipulación de datos en R utilzando SQL

Este material cubre solo la sintaxis de las consultas SQL utilizando el paquete sqldf (https://cran.r-project.org/web/packages/sqldf/index.html) que permite las consultas SQL.

Función sqldf

La función sqldf() generalmente lleva un único argumento que es una instrucción de selección SQL donde los nombres de las tablas son nombres de conjuntos de datos R ordinarios.

```
      library("sqldf")
```

El paquete *sqldf* es simple, desde el punto de vista de R hay una sola función por la que estár preocupados, y es: como se pasa a la función *sqldf()* una instrucción SQL,

```
      sqldf('SELECT age, circumference FROM Naranjo WHERE Tree = 1 ORDER BY circumferenc
```

```
##    age circumference
## 1  118            30
## 2  484            58
## 3  664            87
## 4 1004           115
## 5 1231           120
## 6 1372           142
## 7 1582           145
```

Consultas SQL

Como se ha visto anteriormente en el curso, existe una gran cantidad de comandos principales de SQL, las consultas se realizan con el comando SELECT.

```
      sqldf("SELECT * FROM iris")
      sqldf("select * from iris")
```

Los dos formas de escribir el comandos son equivalentes,

Una nota sobre la convención SQL:

Por convención, la sintaxis de SQL se escribe en MAYÚSCULAS y los nombres de variables/nombres de bases de datos se escriben en minúsculas. Técnicamente, la sintaxis SQL no distingue entre mayúsculas y minúsculas, por lo que puede escribirse en minúsculas o de otro modo. Sin embargo, hay que tener en cuenta que R no distingue entre mayúsculas y minúsculas, por lo que los nombres de las variables y de los data.frame (conjunto de datos) deben tener mayúsculas adecuadas.

El siguiente comando fallaría suponiendo que no se haya creado un nuevo objeto llamado “IRIS”:

```
      sqldf("SELECT * from IRIS")
```

Recordando, la sintaxis básica para SELECT es:

```
      SELECT variable1, variable2 FROM data
```

En el siguiente ejemplo se presenta el comando SELECT para la función sqldf()

```
      BOD
```

```
##   Time demand
## 1    1    8.3
## 2    2   10.3
## 3    3   19.0
## 4    4   16.0
## 5    5   15.6
## 6    7   19.8
```

```
      sqldf('SELECT demand FROM BOD')
```

```
##   demand
## 1    8.3
## 2   10.3
## 3   19.0
## 4   16.0
## 5   15.6
## 6   19.8
```

Nota: SQL no acepta las variables con punto (.) en el nombre, y si lo tiene, debe colocase el nombre de la variable de consulta entre comillas.

```
iris1 <- sqldf('SELECT Petal.Width FROM iris')

# Error: no such column: Petal.Width
```

```
iris2 <- sqldf('SELECT "Petal.Width" FROM iris')
```

COMODÍN *

Se puede especificar un comodín asterisco para extraer todo.

```
      bod2 <- sqldf('SELECT * FROM BOD')
```

LÍMIT

Se utiliza para controlar el número de resultados devueltos

```
      sqldf('SELECT * FROM iris LIMIT 5')
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

ORDER BY

Permite ordenar variables usando la siguiente sintaxis

```
      ORDER BY var1 {ASC/DESC}, var2 {ASC/DESC}
```

Donde la especificación de *ASC* para *ascendente* o *DESC* para *descendente* se realiza por variable.

```
      sqldf("SELECT * FROM Naranjo ORDER BY age ASC, circumference DESC LIMIT 5")
```

```
##   Tree age circumference
## 1    2 118            33
## 2    4 118            32
## 3    1 118            30
## 4    3 118            30
## 5    5 118            30
```

WHERE

Las declaraciones condicionales se pueden agregar a través de WHERE:

```
      sqldf('SELECT demand FROM BOD WHERE Time < 3')
```

```
##   demand
## 1    8.3
## 2   10.3
```

AND y OR

Los operadores AND y OR son válidos para especificar cierto orden de las operaciones en la selección, si se usan juntos es aconsejable separar con paréntesis para especificar el orden.

```
      sqldf('SELECT * FROM rock WHERE (peri > 5000 AND shape < .05) OR perm > 1000')
```

```
##   area     peri    shape perm
## 1 5048  941.543 0.328641 1300
## 2 1016  308.642 0.230081 1300
## 3 5605 1145.690 0.464125 1300
## 4 8793 2280.490 0.420477 1300
```

Hay algunas formas más complicadas de usar WHERE:

IN

WHERE IN se usa de manera similar a %in% de R. También es compatible NOT.

```
      sqldf('SELECT * FROM BOD WHERE Time IN (1,7)')
```

```
##   Time demand
## 1    1    8.3
## 2    7   19.8
```

```
      sqldf('SELECT * FROM BOD WHERE Time NOT IN (1,7)')
```

```
##   Time demand
## 1    2   10.3
## 2    3   19.0
## 3    4   16.0
## 4    5   15.6
```

LIKE

LIKE puede considerarse como un comando de expresión regular débil. Solo permite el comodín % único que coincide con cualquier número de caracteres. Por ejemplo, para extraer los datos donde el feed termina con "bean":

```
      sqldf('SELECT * FROM chickwts WHERE feed LIKE "%bean" LIMIT 5')
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
```

```
      sqldf('SELECT * FROM chickwts WHERE feed NOT LIKE "%bean" LIMIT 5')
```

```
##   weight    feed
## 1    309 linseed
## 2    229 linseed
## 3    181 linseed
## 4    141 linseed
## 5    260 linseed
```

Datos agregados

SELECT pueden crear utilizando los datos agregados AVG, MEDIAN, MAX, MIN y SUM como funciones de la lista de variables para seleccionar. La declaración GROUP BY se puede agregar al agregado por grupos. Se puede nombrar el AS

```
      sqldf("SELECT AVG(circumference) FROM Naranjo")
```

```
##   AVG(circumference)
## 1           115.8571
```

```
      sqldf("SELECT tree, AVG(circumference) AS meancirc FROM Naranjo GROUP BY tree")
```

```
##   Tree  meancirc
## 1    1  99.57143
## 2    2 135.28571
## 3    3  94.00000
## 4    4 139.28571
## 5    5 111.14286
```

Contando datos

SELECT COUNT() Devuelve el número de observaciones. Pasar * o nada devuelve las k totales, pasar un nombre de variable devuelve el número de entradas que no son NA. AS funciona igual de bien

```
      d <- data.frame(a = c(1,1,1), b = c(1,NA,NA))

      d
```

```
##   a  b
```

```
## 1 1  1
## 2 1 NA
## 3 1 NA
```

```
sqldf("SELECT COUNT() as numrows FROM d")
```

```
##   numrows
## 1       3
```

```
sqldf("SELECT COUNT(b) FROM d")
```

```
##   COUNT(b)
## 1        1
```