**SQL**

**Breve historia de SQL**

**SQL** (*Structured Query Language* - Lenguaje de consulta estructurada) es un lenguaje que se basó en 2 principios fundamentales:

1. Teoría de conjuntos.
2. Álgebra relacional de Edgar Codd (científico informático inglés que evoluciono la teoría de conjuntos).

⠀  
**SQL** fue creada en 1974 por IBM. Originalmente fue llamado **SEQUEL**, pero posteriormente se cambió el nombre por problemas de derechos de autor con otra compañía. Mas adelante **SQL** se convertiría en un lenguaje estándar que unificaría todo dentro de las bases de datos relacionales, se convierte en una norma ANSI o ISO; por lo que, la gran mayoría de sub-lenguajes o manejadores de bases de datos relacionales tratan de la misma manera la sintaxis del lenguaje SQL (Oracle, MySQL, PostgreSQL,…).

**Álgebra relacional**

El **álgebra relacional** estudia básicamente las operaciones que se pueden realizar entre diversos conjuntos de datos.

⠀  
No confundir las **relaciones del álgebra relacional** con las **relaciones de una base de datos relacional**.

* Las relaciones en una base de datos se refieren a cuando, básicamente, unes dos tablas.
* Las relaciones en álgebra relacional se refieren a una tabla; o, para ser más precisos, a la interacción que haya con los conjuntos de datos dentro de una misma tabla.

La diferencia es conceptual: Las tablas pueden tener filas o tuplas repetidas pero en el álgebra relacional cada relación no tiene un cuerpo, no tiene un primer ni último *row* (esto debe abordarse de manera más detallada, posteriormente, para su mejor entendimiento).

**Tipos de operadores en Álgebra relacional:**

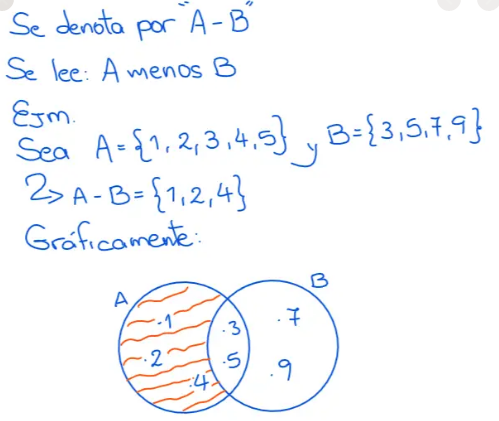
* **Operadores unarios.-** Requieren una tabla o relación para funcionar o correr. *Ejemplos*:

- **Proyección (π):** Haciendo un símil con SQL equivale al comando ***Select***. Saca o extrae, *proyecta,* toda la información de un número de columnas o atributos. Lo dicho, esto sin necesidad de hacer una unión con una segunda tabla o relación. *Simbología de la proyección:* ***π***.  
  
**π**<Nombre, Apellido, Email>(Tabla\_Alumno).  
⠀  
- **Selección (σ):** Haciendo un símil con SQL equivale al comando ***Where***. Consiste en el filtrado de tuplas o filas que cumplan con cierta condición o valor buscado. *Simbología de la la selección:* **σ.**  
  
**σ**<Suscripción=Expert>(Tabla\_Alumno).  
⠀

* **Operadores binarios.-** Requieren más de una tabla para operar (específicamente dos). *Ejemplos*:

- **Producto cartesiano (x):** Toma todos los elementos de una primera tabla, uno por uno, y los combina con todos los elementos de una segunda tabla.   
  
Docentes\_Quinto\_A **x** Alumnos\_Quinto\_A  
⠀  
- **Unión (∪):** Obtiene los elementos que existen, así sea una vez, en cualquiera de las dos tablas; es decir, todo elemento existente, ya sea en tabla A o en tabla B, los une. Une los elementos de tabla A con los de tabla B.  
  
Alumnos\_Quinto\_A **x** Alumnos\_Quinto\_B  
⠀  
- **Diferencia (-):** Obtiene los elementos que existen en una primera tabla seleccionada, pero que no existen en la segunda tabla evaluada en cuestión.  
  
Alumnos\_planExpertPlus **-** Alumnos\_planFree.

***Diferencia al detalle***La diferencia entre dos conjuntos A y B es el conjunto A \ B que contiene todos los elementos de A que no pertenecen a B.



Como ni 7 ni 9 están en el conjunto A, no podemos restar en A esos equivalentes que tenemos en B, pues no los tenemos (7, 9); es decir, se omiten 7 y 9 simplemente en dicha resta. En el conjunto de la “diferencia” se colocan solamente, entonces, los números que no se pudieron restar en A porque no tuvieron su equivalente en B; es decir: 1, 2 y 4.

***Repaso - Propiedad AS***

Usted recordará que existe una propiedad que no es propiamente de ***Select,*** pero que sirve de asistencia a las consultas o ***Queries*** que hagamos con ***Select;*** esa propiedad es: ***AS.***

***AS*** nos permite pasarle un *alias,* o nombre temporal *que no sustituye de manera permanente al nombre original,* a una o varias columnas (o incluso tablas); es decir, ***AS*** le pasaría un nombre temporal, cambiando temporalmente el nombre original, a una o varias columnas seleccionadas (o tablas) a la hora de querer hacer una consulta de ellas, como se esperaría, con ***Select***. Es decir, ***AS*** nos permite sustituir temporalmente, por lo menos en el justo momento de hacer propiamente una consulta, el nombre original de una o de varias columnas o tablas; de tal manera que se le asigne un nombre temporal a la columna (o tabla) de nuestro interés.

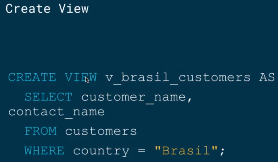
El proceso es más o menos así: por medio o no de ***Select,*** se llama a la *o a las columnas o tablas* de nuestro interés con su nombre original (Si las llama sobre la misma línea de código, no se olvide hacer separaciones por coma por cada columna o tabla llamada)**;**posteriormente, se le debe pasar a cada nombre por separado, e inmediatamente después del nombre de cada columna o tabla llamada, la sentencia ***AS;*** entonces, luego de declarar ***AS,*** se le pasaría un nuevo nombre (que sería el *Alias*), el de nuestro interés, a cada columna o tabla llamada en cuestión,... tal que así, por ejemplo:



Advertir que lo único que cambia, del *dataset,* y de forma *temporal,* es el nombre original al hacer una consulta con ***Select;*** del resto, todo sigue igual, los mismos datos para las mismas columnas o tablas: sólo que ahora está viendo que esas columnas o tablas afectadas están siendo llamadas con otro nombre; los cuales, fueron pasados con la propiedad ***AS.***

**Ahora, algo interesante.** En adelante, de hecho, usted podrá llamar cualquiera de las dos formas a sus columnas de interés para manipular sobre ellas, ya sea que las llame con su nombre original o por medio del *Alias* recientemente pasado. Por ejemplo, en futuras ***Queries*** más complejas puedes referirte a los campos en cuestión, ya no con su nombre original; sino, también, con su *Alias* asignado (si así lo deseas).

**Ahora, ponga atención a lo siguiente:** **AS**, como instrumento, **no siempre se usa como propiedad que asiste a la sentencia *SELECT* para definir un *alias* posterior a su declaración*,* No. A veces, y en los casos puntuales en los que se crean *VIEWS,* el alias o nombre que recibirá la vista en cuestión, de hecho, se define de primero y antes de hacer uso del recurso de *AS* (antes de declarar AS)*.***Ahora sí,justo después de declarar AS, en VIEWS, no se establece un *alias*, NO; si no que de entrada se establece qué información deseamos reflejar y filtrar dentro de la View en ejecución. ***Tal como lo hicimos en este ejemplo, observe:***

 **En este *View* se pretende reflejar la información de la gente que viene de Brasil, puntualmente, la información de su nombre y de su contacto.**

El anterior ejemplo lo estuvimos viendo en el documento: ***1. SQL DDL,***de la carpeta *Platzi,* explicado entrelas páginas 6 & 7 (por si desea saber más sobre el Objeto ***View***).

***Repaso - Propiedad Count***

***Select,*** también, nos permiten crear datos de manera temporal que nos pueden agregar información adicional; un buen ejemplo de eso sería combinar el ***Select*** con el uso de la sentencia ***Count(),*** ¿y qué hace ***Count()***?

La sentencia ***Count()*** se encarga de contar, ¿contar qué? Contar ***Registros.*** Es decir, a partir de unos parámetros pasados para ***Count(),*** éste últimose encargará de contar la totalidad de los registros que hay dentro de una tabla, si y sólo si, se cumplen a cabalidad unas condiciones o parámetros pasados ¿qué parámetros? Pues, en principio, los que establecemos dentro de los paréntesis de ***Count().***

El ejemplo más básico de uso de la sentencia ***Count(),*** en connivencia con la sentencia ***Select,...*** es que en ***Count()*** se le pase, como parámetro, que cuente absolutamente todos (sin filtro alguno) los registros existentes para una tabla seleccionada con la sentencia ***From***; entonces, en este caso, ese “nuevo dato que se crearía temporalmente y que nos podría agregar más información” sería el dato numérico, que se nos arroja, equivalente a la totalidad de registros que se alcanzaron a contar para una tabla seleccionada en cuestión.

Pongamos, por ejemplo, el siguiente caso: 

Esto nos arrojaría un nuevo dato, un simple número, que reflejaría el conteo total de los *registros* que tenemos en nuestra tabla *posts;* es decir, prácticamente, la cantidad de *posts* que tenemos publicados. Tal que así:

***, 22*** sería el número de *registros,* o posts, que tenemos en nuestra tabla *posts* parcialmente.

Ahora, si se da cuenta, ese nuevo dato que se ha creado, el dato numérico, está sustentado por un nuevo atributo, campo o columna que también ha sido creado; esa nueva columna si se percata se llama: ***COUNT(\*)***

Luego, se dará cuenta que, a la nueva columna resultante ***COUNT(\*)*** se le puede pasar un *Alias* tambiéncon la sentencia ***AS;*** esto para recordar o renombrar, a dicha columna, de una manera más fácil e *identificativa*; pues, es *engorroso* tener que llamar a una columna con una palabra clave como ***Count(\*****).* Probemos:



Esto quiere decir que, nuestra columna inicialmente llamada ***COUNT(\*),*** ahora puede ser llamada también como: ***numero\_posts***; lo cual, tendría mucho más sentido ya que ***numero\_posts***, como nueva columna, nos estaría mostrando justamente el número de postsque tiene la tabla ***posts***,... Quedando así:



Por supuesto, hay que decir que realmente este dato no está almacenado de forma permanente en la base de datos. Con todo y eso, lo podemos aún así consultar por medio de ***Select.***

***Un abreboca…***

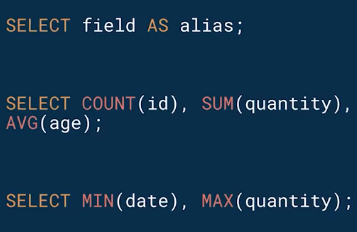
***Funciones de agregación***

Las funciones de agregación en SQL nos permiten efectuar operaciones matemáticas sobre un conjunto de registros evaluados, o seleccionados para ser evaluados, dentro de un campo o columna (generalmente de tipo numérica) determinada. Al efectuar este tipo de operaciones matemáticas se devuelve un único valor agregado, como resultado, en el análisis de los valores evaluados o fijados a evaluar. *Tome como referencia la propiedad* ***Count*** *previamente recordada.* Sin embargo, a lo sumo, podríamos incluso calcular ***medias, máximos, sumas***,… sobre dicho conjunto de valores determinados; estos valores se determinan por condiciones o parámetros pasados que tienen mucho que ver con fijar qué columna(s) y/o fila(s) será(n) evaluada(s).

*Las funciones de agregación básicas que soportan todos los gestores de datos son las siguientes:*

* **COUNT**: devuelve el número total de filas seleccionadas por la consulta, como particularidad se puede usar COUNT()\* donde contará todos los registros de la tabla, incluyendo nulos.
* **MIN**: devuelve el valor mínimo del campo que especifiquemos.
* **MAX**: devuelve el valor máximo del campo que especifiquemos.
* **SUM**: suma los valores del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.
* **AVG**: devuelve el valor promedio del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.

***Modelación***



***Función IF()***

Estructura de control que se encarga de evaluar la validez lógica de una expresión o condicional definida por medio de operadores relacionales. Según el resultado lógico que arroje la condición evaluada, ya sea verdadera o falsa, se mostraría por pantalla lo que fue definido como salida verdadera o, en el caso contrario, como salida falsa, respectivamente.

IF (expresión lógica, resultado/salida\_true -de ser la expresión verdadera-, resultado/salida\_false –de ser la expresión falsa-).

***Función CASE()***

Es también una estructura de control, en principio mutuamente excluyente, que sirve para evaluar la validez lógica de una lista de expresiones y/o condicionales (no sólo de una). En caso de que alguna expresión lógica de este listado sea valida (y sólo una debe ser válida por ser mutuamente excluyentes) se retornaría, por pantalla, su salida o valor definido como verdadero.

La sintaxis comienza con la sentencia CASE, luego se evalúa la validez lógica de las expresiones valiéndose de la propiedad WHEN; y, en caso de encontrar valida a alguna expresión lógica, dicha expresión devolverá por pantalla la salida especificada como verdadera para su propia condición evaluada (lo que se refleja por pantalla, como verdadero, es justo lo que se coloca después de la propiedad THEN).

La clausula ELSE, que complementaria a CASE, es opcional. ELSE devolverá su salida o valor definido como verdadero en caso que todas las condiciones WHEN, anteriores, hayan sido falsas.

Ahora, si todas las condiciones son falsas, pero tampoco existe la clausula ELSE, se devolverá NULL.

La sentencia CASE es finalizada con la palabra clave END. Sin embargo, a las sentencias CASE también se les puede asignar un *alias*. De hecho, al ejemplo a continuación se le asignó uno.

***Modelación***

