**Introducción a Consultas a bases de Datos**

**Sobre la importancia de las consultas (o Queries) en una base de datos relacional.**

Las consultas son la parte fundamental de una base de datos. De qué te sirve hacer una consulta o un Query efectivo?

Pues bien, una consulta bien hecha podría, incluso, *salvar un negocio o una empresa.* Las empresas tienen una gran necesidad por información correcta, exacta, oportuna; dicha información, se puede extraer generalmente de las bases de datos que ya tenemos (de aquí la importancia de las *consultas*).

En las bases de datos tenemos muchas veces *clientes, productos, proveedores* y diferentes tipos de entidades que, por separado, no tendrían mucho sentido; pero, cuando se comienzan a unir, a través de Queries, puedes presentar: un informe, o una tendencia, etc...; de tal forma que, las personas que están encargadas de *maniobrar* la empresa para un lado o para el otro,... tengan la información que necesitan para tomar las decisiones correctas.

Entonces, alrededor de toda esa cultura de “*extraer y transformar información”,* se han creado varias especialidades en el esquema de datos que tiene que ver con, por ejemplo, ***ETL*** (transformación de datos), ***Business Intelligent*** (extracción & utilización de datos para la toma correcta de decisiones) e, incluso, que tiene que ver con: ***Machine leearning*** (toma los datos existentes de una base de datos y los presenta de tal manera que haga sentido, que sea coherente y, sobre todo, que sea útil).

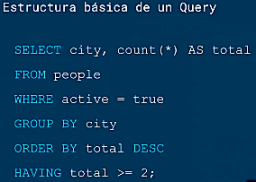
*En resumen, los Queries son una parte fundamental de una base de datos;* pues, nos permiten conocer las estructuras de los **Query** y resolver cualquier duda que tenga cualquier miembro de la empresa que pueda ser trabajada a través de una base de datos.

En resumen, con los ***Queries*** es que se brinda valor a una organización al final del día; pues, es lo que nos trae la información, *el diario vivir dentro de una organización*; con lo cual, es que se podrán tomar decisiones de gestión e inversión diariamente. Entonces, para traer toda esa información de nuestro interés, de una base de datos, necesitas hacer ***Queries.*** Los ***Queries*** son la forma en la que estructuramos las preguntas que le vamos a hacer a la base de datos; es decir, al final del día, nuestro camino a seguir para encontrar todas las respuestas.

Ahora, lo complicado es que hay muchas formas de hacer un ***Queri,*** desde lo más básico a lo más complejo.

***Vamos de las Queries básicas a las complejas: Select***

Empecemos a practicar la habilidad de convertir nuestras preguntas en ***Queries.*** El ***Queri*** tiene básicamente dos partes: la sentencia ***Select*** y la sentencia ***From*** y, en varias ocasiones, nos encontramos con la sentencia ***Where.***



***Select*** nos permite traer los datos que queremos mostrar, todos ***(\*)*** o de alguna ***columna*** en especifica; en este caso puntual, queremos traer los datos de la columna o atributo ***city*** y una cuenta ***count(\*)*** la cual se proyecta como ***total*** (en adelante explicaremos más a fondo los dos últimos elementos señalados en negrilla).

*Un dato:* En caso que usted no sólo desee ver los datos de una única columna, de una tabla, puede agregar más. Simplemente, dentro de la misma sentencia ***Select,*** debe llamar al otro u otros campos en cuestión por medio de separaciones de coma; por ejemplo: ***SELECT city, name*** (Ahora veríamos, ademas de ***city,*** la columna ***name).***

En ***From*** indicamos de dónde vamos a seleccionar esos datos que queremos mostrar con el ***select;*** es decir, más precisamente, de que ***tabla***; en contexto, entonces, estamos indicando que queremos los datos de la columna o atributo ***city*** de la tabla ***people.***

Ahora, en ***Where,*** el filtro de la consulta y recolección de datos se vuelve más exigente; pues, con ***where,*** lo que se establece es una condición; en consecuencia, dicha condición debe cumplirse para poder seleccionar los datos deseados de una columna para una tabla en especifica. En contexto, cuando definimos un where con ***active = true*** le estás diciendo a nuestra base de datos de trabajo actual que desea solamente los datos de la columna ***city*** de la tabla ***people*** que cumplan con la condición ***active = true.*** De hecho, ***active*** vendría siendo también otra columna o atributo de la misma tabla ***people*** y ***true*** es uno de los posibles valores que puede tener, dentro de dicha tabla, la columna ***active.***

***Group by*** lo que nos permite es agrupar los datos, justamente, según un criterio pasado; en este caso, el criterio pasado fue: ***city;*** es decir, vamos a agrupar los datos de la tabla ***people***, que cumplen con la condición ***active = true*** y son de la la columna o atributo ***city,*** dentro de la misma columna ***city*** (nos mostrará la info de ***city***).

***Order by*** define de qué manera se organizarán los datos recolectados; en este caso, nos ayuda a definir de qué manera ordenar los datos seleccionados de la columna ***city***, de la tabla ***people***, que cumplen con la condición ***active = true*** (la totalidad de ellos); por ejemplo, definir el orden de los datos seleccionados de tal manera que: se *enlisten de mayor a menor* las ciudades ***(city)*** según la cantidad de personas que tenga cada una.

Por último, en ***Having,*** sedefine otro parámetro de limitación, va a aplicar otro filtro o condición mucho más especifico; en este caso, va a filtrar (elegir) el ***total*** de los datos seleccionados de la columna ***city***, de la tabla ***people*** y que tengan un ***active = true;*** pero, además, *cuando haya 2 o más de 2 personas en cada ciudad* ***(total >= 2).*** Tengo la sensación que ***total*** es otra columna o atributo de la tabla ***people.***

***un dato:*** La sentencia ***Select \**** significa *selecciona todos los datos;* sin embargo, qué es *todo?* SQL interpreta esta solicitud, por sí sola, como muy ambigua y tiene razón (por lo cual te lanza un error). Es decir, a ***Select \**** se le debe acompañar en todos los casos con otra sentencia. Por ejemplo, evaluemos nuestro primer caso de ***Query:***



En efecto, con esto estamos diciendole a la base de datos que nos traiga absolutamente todo ***(SELECT \*)***; pero, recuerde, falta algo más,... Queremos todo, sí, pero de la tabla ***posts (FROM posts),*** que nos muestre absolutamente todo de ahí (sus campos, columnas o atributos).

De hecho, generalmente, te encuentras las ***Queries*** con una tercera sentencia declarada que es, como te contabamos, ***Where;*** que nos permite filtrar cuáles de esos datos, de la tabla ***posts***, queremos ver.



En este caso, sólo hemos filtrado los datos de la tabla ***posts*** que tienen como fecha de publicación una fecha igual o superior al año ***2024.*** Es decir, la base de datos nos arrojará todas las noticias, ***posts,*** que sean del ***2024*** para ***adelante*** en su ***fecha\_publicacion.***

***Nota:*** *para SQL “>” es igual a “>=”, por lo que en el caso anterior se incluye también al año 2024. Sin embargo, “<” no es lo mismo que “<=”.* **En resumen, con *Queries* y *Select,* filtramos información.**

En adelante, probaremos ***Queries*** más complejas según el nivel de complejidad que tenga, propiamente, la consulta de los datos que se desean ver; que, obviamente, deberían de ser en la medida unos datos con unas caracteristicas más puntuales o con mayor especificaciones. Sin embargo, para hacer consultas, se requiere tener primeramente datos; dicho eso, empecemos por agregar datos de prueba a nuestras tablas de modelación.

***Propiedad AS***

Ahora, 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; esto, si se acompaña con la sentencia ***Select***; es decir, ***AS*** le pasaría un nombre temporal, cambiando temporalmente el nombre original, a una o varias columnas seleccionadas a la hora de querer hacer una consulta de ellas con ***Select***. Es decir, ***AS*** nos permite sustituir temporalmente, por lo menos sólo en el momento de hacer propiamente la consulta, el nombre original de una o de varias columna; de tal manera que se le asigne un nombre temporal a la columna (o columnas) de nuestro interés.

El proceso es más o menos así: Por medio de ***Select,*** se llama a la *o a las columnas* de nuestro interés con su nombre original (no se olvide hacer separaciones por coma por cada columna llamada)**;**posteriormente, se le debe pasar a cada nombre por separado, e inmediatamente después del nombre de cada columna 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 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: sólo que ahora está viendo que esas columnas 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).

***Propiedad Count***

***Select,*** también, nos permite hacer datos que se crean en el momento y que nos pueden agregar información; 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 parametros 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 establecidas según dichos parametros pasados en cuestión, ¿qué parametros? Pues, en principio, los que establecemos dentro de los parentesis 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 parametro, 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 en el momento y que nos podría agregar información” sería el dato númerico, que se nos arroja, equivalente a la totalidad de registros que se alcanzarón a contar, en su totalidad, 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 númerico, 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 identificar, a dicha columna, de una manera más facil 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 pues ***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.***

***From + Join***

Como usted ya sabe, por cuestiones de *normalización,* siempre se busca que cada entidad se estudie por separado; esto, al asignarles tablas independientes, aunque relacionadas, a cada una de ellas. Sin embargo, a la hora de presentar un informe, a la hora de traer información que sea valiosa, conviene juntar dichas tablas independientes; pues, nunca dejan de relacionarse y nos ayudaría, justamente, a *desarrollar información.*

En consecuencia, para lograr ese *junte,* como parte de la sentencia ***From,*** tenemos un compañero inseparable de ella***,...*** es la sentencia ***Join.*** Entonces, con ***From*** no sólo se selecciona de qué ***tabla*** queremos visualizar unos datos; sino que, asistiendose de ***join,*** también podríamos unir varias tablas entre sí a través de las relaciones que les asignamos en un principio; es decir, más concretamente, uniendo llaves foráneas con llaves primarias en caso de existir dicho vinculo. Eso es lo que veremos ahora.

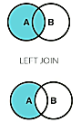
Los ***Join*** tratan o parten de una base matematica relacionada a la *teoría de conjuntos;* esto lo veremos, de una forma gráfica, con algo que se llama: ***Diagramas de Venn. Los Diagramas de Venn*** son simplemente circulos que se tocan, en algún punto, para ver dónde está la intersección de conjuntos y, en últimas, ver dónde hay un traslape de valores de un lado y del otro. Veamos esto.

***Diferentes tipos de Join***

**1. Diferencia:** Cuando hablamos de“Diferencia” nos referimos a lo que tenemos en un lado, ignorando lo que está en el otro (se seleccionan los datos que están en una tabla ***A***, por ejemplo, y se excluyen los datos que están en una tabla ***B***); es decir, hay un ***Join*** que se encarga de mostrarnos solamente la información que tiene cada tabla por aparte (el ***Join*** de la ***Diferencia***). Ahora bien, si quisieramos ser más exigentes en nuestra toma de datos con el tipo de ***Join “Diferencia”***; de nosotros depende si queremos excluir, también, los datos de la tabla ***A***, continuando con el mismo ejemplo, que se relacionen o se repitan en la tabla ***B***.

O sea, quiero que quede claro, uno podría hacer una ***diferencia*** al tomar la totalidad de los datos de una tabla ***A***, sin importar si uno de sus datos se relacione o se repita en tabla ***B***; o, bien, siendo más rigurosos, podríamos tomar la totalidad de los datos de una tabla ***A***; pero, excluyendo aquellos datos que se repitan en tabla ***B;*** es decir, ignorando la intersección entre los conjuntos,***...*** En cualquiera de los dos casos, sepa que se estaría hablando de ***Left Join*** (si se supone a priori que la tabla ***A*** está situada en la parte izquierda y la tabla ***B*** está situada en la parte derecha).

Tal que así:



Ahora, suponiendo que la tabla ***A*** estuviera situada en la parte derecha y la tabla ***B*** en la parte izquierda,... estaríamos hablando de ***Diferencia*** de ***Right Join.*** Por suerte conservamos cada tabla en su mismo lado:



**2. Intersección:** Cuando hablamos de “Intersección”, nos referimos a los datos que tenemos en común dentro de cada tabla, los datos que se repiten, que están están en ambas tablas; es decir, más puntualmente, estamos hablando de seleccionar los datos de la intersección entre los conjuntos. Si retomamos el ejemplo anterior estaríamos hablando de seleccionar los datos que están tanto en la tabla ***A*** como en la tabla **B**. Por lo general, este es el tipo de ***Join*** que más se ve y el nombre correcto que recibe es: ***Inner Join*** o en español ***Join interno.***



**3. Unión:** Cuando hablamos de “Unión”, estamos definitivamente hablando de unir ambas tablas, tanto ***A*** como ***B,*** por ejemplo; es decir, acá se seleccionarían y se unirían la totalidad de los datos, sin excepción alguna, de ambas tablas. *Unión* trae absolutamente todos los datos de ambas tablas.



***4. Diferencia simétrica:*** Acá también se tomarían los datos de ambas tablas,... menos los datos que se repitan en cada una de ellas; es decir, los datos que tengan en común ambas tablas, tanto la tabla ***A*** como la tabla ***B,*** por ejemplo. En resumen, se tomarían todos los datos de ambas tablas; pero se omitirían, solamente, los datos de la intersección entre los conjuntos. Se ve así:



***Ahora sí, veamos cada uno de los 4 ejemplos por medio de código SQL en MySQL Workbench,... Le daremos sentido a todo esto.***

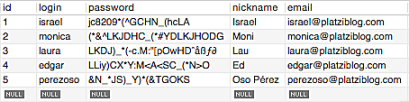
***Utilizando la sentencia FROM en MySQL***

***Poniendo en práctica Left Join & Right Join: bajo este tipo de Join uniremos “usuarios” con “posts”.***

Primero que todo nos interesa efectuar la práctica de ***FROM*** en la tabla ***Usuarios***; pero, antes que nada, visualizemos toda la tabla ***usuarios,*** en su conjunto, corriendo las siguientes sentencias que ya conoce:



Se nos *muestra*, como resultado, lo siguiente:



***Left Join.***

***Si hacemos un simil con los Diagramas de Venn, “usuarios” será “tabla A” & “posts” será “tabla B”;*** entonces, en ese orden de ideas, la tabla que usted seleccione de primero será instantaneamente la de la *parte* *izquierda,...* adicionalmente, debajo de ***FROM usuarios,*** se ha declarado la sentencia ***LEFT JOIN***,... con eso estamos afirmando que la prioridad la tendrá la tabla ***usuarios***; es decir, la ***tabla izquierda,*** la ***tabla A*** (Recuerde, la 1ra tabla que se pase siempre será la tabla del *lado izquierdo* o *tabla A*)***.*** Entonces, como la ***tabla A*** tendría la prioridad, de la unión resultante con la ***tabla B***, ***los datos de la tabla A se verían primero; y, de hecho, se verían en su totalidad,... es decir, se verían todos los usuarios independientemente de si los usuarios hayan escrito o no posts. Esto último es lo más importante. Y acuerdese siempre de la relación que tengan ambas entidades; pues, serán fundamentales para las conclusiones que podríamos tener para las Queries que hagamos.***

Sin embargo, retomemos, este conjunto de sentencias está incompleta; recuerde que su intención es, precisamente, hacer un tipo de ***Join*** que es de ***Diferencia,*** al tratar de unir una ***tabla A*** con una ***tabla B***; entonces, para señalar cúal pasaría a ser esa ***tabla B***,... se le pasa el nombre de la segunda tabla de su interés, en este caso ***Posts,*** justo al lado de la sentencia ***LEFT JOIN.*** Tal que así:



Ahora, ya dejamos claro qué tablas vamos a unir y hemos decidido, entre las dos, cual será ***tabla A*** y cual será ***tabla B,*** y que la unión será de tipo ***LEFT JOIN*** entre ambas; sin embargo, hace falta especificar la manera en cómo lograremos que ambas tablas, en definitiva, se unan; y, eso se logra, ***conectando la llave primaria de la tabla A***, ***usuarios,*** que es la tabla de nuestra prioridad por ser la tabla de ***la parte*** ***izquierda*** y por hacer la unión desde un ***left join,...*** ***con la llave foranea de la tabla B*** que, justamente, se relaciona o viene importada de la llave primaria de ***usuarios.*** Lo anterior se logra al *igualar* ambas columnas, la de la llave primaria de ***usuarios*** con la de la llave foranea de ***posts*** (relacionada a usuarios), por medio de la palabra reservada **ON.** Se *igualan* porque se supone que, en efecto, sus datos son iguales entre sí. Veamos esto en código:



En ***usuarios.id, “usuarios”*** hace referencia a la tabla como tal, luego hay una separación de palabras por medio de un punto (.), e ***“id”*** hace referencia, ahora sí, a un campo de la tabla ***usuarios;*** que, como ya debe saber, es el campo correspondiente a la llave primaria de dicha tabla ***usuarios.***

Ahora, en ***posts.usuario\_id, “posts”*** hace referencia a la tabla como tal, luego hay una separación de palabras por medio de un punto (.), y ***“usuario\_id”*** hace referencia, ahora sí, a un campo de la tabla ***posts;*** que, como ya debe saber, es el campo correspondiente a una llave foránea (que se relaciona con la llave primaria de la tabla ***usuarios)*** de dicha tabla ***posts.***

En resumen, las tres sentencias conjuntas, al ser ejecutadas, nos dice lo siguiente:

***Selecciona todo, de la tabla “usuarios”, y unela con la tabla “posts”; reflejando primero la tabla “usuarios”, por ser la de la izquierda, y luego la tabla “posts”, por ser la de la derecha. Dicha unión se logra por medio de las dos llaves que las relaciona; concretamente, para esto, “usuarios.id” debe ser igual a “posts.usuario\_id” (como se supone).***

Con esto tenemos los datos, unidos, de ambas tablas: ***usuarios & posts.***

Ahora, si queremos saber puntualmente que usuarios NO han escrito ningún post; recuerdese de la sentencia ***WHERE*** que nos ayuda a, precisamente, buscar un dato o unos datos en especifico, de una tabla pasada (o, en este caso, de la combinación de dos tablas por medio de sus llaves en común, una propia y otra foránea), según un filtro declarado; en esta ocasión el filtro es: ***querer saber qué posts están vacios y a que usuario le corresponde;*** más concretamente, donde las celdas de la tabla ***posts*** estén vacias o ***NULL*** para x o y ***usuario.***

Como en este caso puntual estamos haciendo busquedas especificas con la sentencia ***WHERE;*** pero, **sobre dos tablas unidas**; entonces, para respetar y mantener dicha unión entre ambas tablas, a ***WHERE,*** hay que pasarle la llave foranéa de la tabla ***posts****;* no cualquiera, sino, la que se vincula con la tabla ***usuarios*** justamente***.*** Ahora, uno habla de ***posts*** porque el filtro se aplica es sobre dicha tabla, no sobre otra,... se pretende buscar sus campos vacios o ***NULL***; y, principalmente, por lo que ya dijimos, porque es ***posts.usuario\_id*** la llave que relaciona a ambas tablas al fin; pues, ya están unidas y se debe respetar esa unión,... realmente cualquier ***NULL*** que encuentre, entre una tabla o la otra, lo arrojará (mostrará). Como en este caso la prioridad la tiene la tabla ***usuarios,*** se mostrarán todos los datos de los ***usuarios*** registrados; por lo que, entonces, se espera ver los ***NULL*** en los campos de la tabla ***posts.*** La declaración de la sentencia ***WHERE*** queda así:



y ahora todo junto se vería así:





***En el caso contrario, Right Join...***

La ***tabla B*** tendría la prioridad de la unión resultante con la ***tabla A***; es decir, ahora, ***los datos de la tabla B se verían primero que los de la tabla A; y, de hecho, ahora se verían en su totalidad los datos de la tabla B; esto, en otras palabras, significa que se verían todos los posts,... independientemente de si haya registro o no del usuario que lo o los escribió.***

Como usted ya sabe, la 1ra tabla pasada siempre será la ***tabla A*** o tabla de la ***izquierda***; es decir, eso no va a variar ahora. Lo único que cambiará ahora es la declaración de la sentencia ***LEFT JOIN*** por la de ***RIGHT JOIN,...*** para darle la prioridad a la ***tabla B,*** a la tabla de la ***derecha,*** en este tipo de ***Join (y ya sabe lo que significa eso)***





Por lo general las tablas se unen para hacer *filtros* entre ellas según la relación, de cardinalidad, que ellas tengan. Recuerdese que la cardinalidad entre ***usuarios*** y ***tablas*** era de ***1:N*** y la ***relación*** era ***usuarios escriben posts.*** Base sus ***joins*** de tablas, y los posteriores *filtros* que haga para ambas, al estar unidas, según la relación que conserven (como lo hicimos, justamente, con ***usuarios y posts***).

***Poniendo en práctica Inner (Intersección): bajo este tipo de Join uniremos, ahora, “usuarios” con “posts”.***

***Inner***

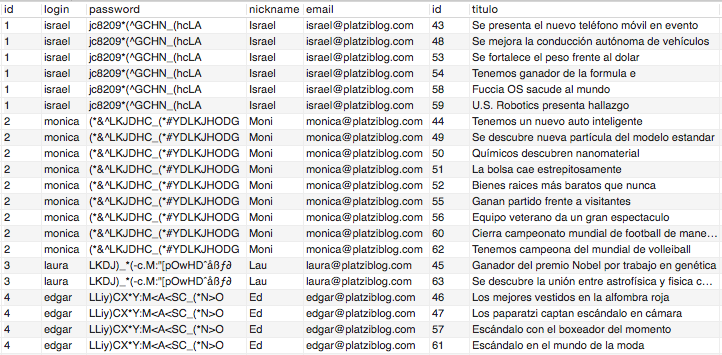


La ***intersección*** tiene que ver con que, recordemos, los datos de cada registro de ambas tablas (a & b) se correspondan entre sí y, en consecuencia, no se encuentren celdas vacías al unirse las tablas en el ***query;*** es decir, acá al hacer la consulta, no se estaría llamando todas celdas con información de una única tabla A, por ejemplo, sin importar si cada una de sus celdas se encuentre correspondida, en términos de información (no estén vacias tampoco), con las celdas de la tabla B; o, por el contrario, no se estarían llamando tampoco todas las celdas con información de una única tabla B sin importar que sus celdas estén correspondidas, en términos de información, con las celdas de la tabla A, NO. Lo que estamos pidiendo en la consulta, ***query,*** es que veamos todas las celdas, tanto de la tabla A como de la tabla B, cuando ninguna de sus celdas estén vacias en cada uno de los ***registros.*** Entonces, esta es la condición: las celdas, en cada uno de los registros para ambas tablas, no pueden estar vacías para que el ***query*** las pueda mostrar; de lo contrario, el ***join inner*** no las mostrará. En contexto, ***join inner*** sólo nos mostrará los ***usuarios*** con necesariamente registro en ***posts*** y los ***posts*** con necesariamente registro en ***usuarios.***



Acá se respeta el mismo orden en el que son nombradas las tablas. La ***tabla A*** o ***de la izquierda,*** manteniendo el simil con el diagrama de Venn, será la primera tabla llamada; es decir, ***FROM*** ***usuarios.*** Luego, la tabla B o de la derecha será la segunda llamada.

La palabra ***INNER JOIN*** quiere decir: *“sólo traeme lo que está internamente ligado, traeme solamente los que tienen dependencia de ambos lados”* (donde no haya celdas vacías en tabla A ni en tabla B).



Este es uno de los ***Joins, Inner Joins,*** de los que más se van a utilizar y es, generalmente, el que más valor brinda; pero, hay muchas veces que es necesario utilizar ***Left*** o ***Right*** para que no te importe uno de los lados y siempre traigas todos los datos de un lado de la ecuación.

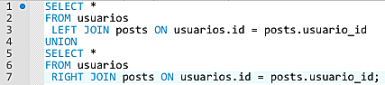
***Poniendo en práctica Unión: bajo este tipo de Join uniremos, ahora, “usuarios” con “posts”.***

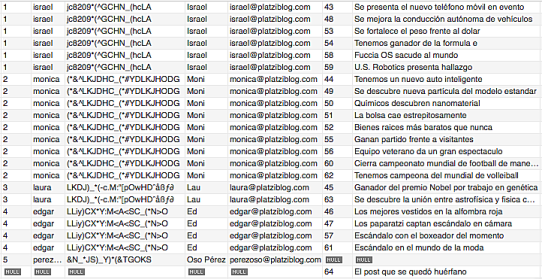
***Unión***



Este ***Join*** es muy claro. Como se dijo antes, cuando hablamos de *Unión,* estamos hablando de unir definitivamente ambas tablas, tanto ***A*** como ***B,*** sin importar si en los *registros* de cada una de las tablas hay celdas *vacías.*

Este tipo de ***join,*** en algunos manejadores de bases de datos, no tiene una forma propia de sentencia para ser ejecutado; por lo que el *estandar* es que sea ejecutado al *unir* las sentencias ***left join*** (que muestra la totalidad de los registros completados de *la tabla izquierda* o tabla A con el registro correspondiente en tabla B; sin importar si las celdas, de cada registro de la tabla B, estén vacías o no) ***& right join*** (que muestra la totalidad de los registros completados de *la tabla derecha* o tabla B con el registro correspondiente en tabla A; sin importar si las celdas, de cada registro de la tabla A, estén vacías o no); evidensiandose así, a lo sumo, por medio de dicha unión, la totalidad de las celdas de cada tabla, estén vacías o no. De hecho, si tú unes los diagrams de Venn del ***Left join*** con el de ***Right join*** te dará ***Unión.*** Es decir, en contexto, ***se verían todos los usuarios independientemente de si los usuarios hayan escrito o no posts; pero, también, se verían todos los posts independientemente de si haya registro o no del usuario que lo o los escribió.*** En código se vería así:





***Por último, pongamos en práctica Diferencia simétrica: bajo este tipo de Join uniremos, ahora, “usuarios” con “posts”.***

***Diferencia simétrica***



Sin extendernos mucho, básicamente consiste en: ***ver lo que existe en A pero no en B y ver lo que existe en B pero no en A* (es justamente todo lo contrario a *Inner join*).** Tal que así:



