PostgreSQL 9.1: consultas SQL sencillas con phpPgAdmin.

Aprovecho el breve receso de estas fechas carnestolendas de 2015 para continuar con esta miniserie de artículos sobre PostgreSQL 9.1. Para aquellos que quieran llevar la secuencia correcta del tema éste se inicia acá .

La fotografía que veís y que encabeza esta entrada es la del Doctor Donald Chamberlin, investigador de la IBM quien junto al Doctor Raymond Boyce (quien falleció en 1974 y muy poquísimas fotografías dejó) ambos idearon (y nos legaron) el "Structured Query Language" (SQL).

Lo que haremos hoy son unas consultas sencillas en SQL no sin antes haber agregado unos cuantos datos de prueba (**cualquier semejanza con personas reales es pura y casual coincidencia**).

Utilizando el navegador web predeterminado en *Debian*, el *Iceweasel* (que cada día aprecio más por su economía de recursos y por ende rapidez) nos vamos a la consabida dirección IP de nuestro servidor virtual:

Introducimos el "login" y la contraseña (si no lo recuerdan ir a esta entrada) y una vez hayamos logrado acceso hacemos click ("o vamos") a la tabla *students* (aunque soy fanático del castellano, debo reconocer que el inglés es hoy en día lo que fue el latín en el mundo entero) y observamos su estructura de datos más sin embargo nos interesa, por ahora, el enlace "insert" para dar algunas altas en la tabla:

Tras lo cual se nos abre el siguiente cuadro de diálogo el cual procedemos a llenar sin cortapisa:

Observen que dejé el campo "id" intencionalmente en blanco: nuestro primer valor será, naturalmente, el 1 pero en los siguientes registros que insertemos intenten guardar ese valor de nuevo y observarán una de las restricciones de la tabla: los valores no pueden repetirse para ese campo en particular. A la final verán algo parecido a esto, ya con los registros insertados:

Vuelvo a repetir: cualquier parecido con los datos personales en la realidad es simple y feliz coincidencia. Ahora que tenemos unos cuantos registros insertados podemos hacer unas consultas sencillas, para ello bastará con hacer click en el enlace "SQL" que está en el borde superior derecho de la web, cerquita de

"SQL | History |Find | Logout"

¿lo vieron? en el cuadro de diálogo que se abre escribimos la siguiente sentencia para visualizar los correos electrónicos de los estudiantes:

```
select email from students;
```

Antes que hagan click en "execute" ("ejecutar -sentencia-") observen atentamente que estamos trabajando con la base de datos "GNU_academy" y que nuestro vía de búsqueda de esquema está apuntada a "public" (ésto último permite que al ejecutar una sentencia SQL los nombres de tablas y campos sean buscados y verificados primero antes de comenzar a buscar datos, e incluso es útil para asuntos de seguridad y privilegios pero ese tema es más avanzado; si luego quieren aprender más sobre ello aquí el enlace introductorio al asunto).

Así pues, sin más, ejecutamos la sentencia tras lo cual veremos algo parecido a esto:

La explicación breve de la sentencia SQL es traducirla del inglés:

"select"->"selecciona"

"from"->"desde (la tabla)"

";"-> indica fin de la sentencia, **no es obligatorio** pero es útil si vamos a introducir varias sentencias que deben ir en un orden específico (por ejemplo insertar más datos de estudiantes y luego mostrarlos en un listado para asistencia -ya le vamos dando utilidad en la vida real a esto de las bases de datos-).

Como hay comandos específicos del lenguaje SQL yo opto (y hay varias personas que lo hacen) por escribir dichos comandos en mayúsculas y los nombres de tablas (y/o variables y/o constantes) en minúsculas y como pueden ver se ejecutan sin ningún problema, el asunto es de legibilidad para nosotros los humanos:

```
SELECT email FROM students;
```

Importante: los nombres de campos y tablas han escribirse tal cual fueron creados en la base de datos correspondiente, de no ser así nos devuelve error; es por ello que desde un principio en estos tutoriales los creamos todos en minúsculas -y en inglés, de paso-.

Volviendo al tema de la realidad, supongamos que por cualquier razón necesitamos saber cúales (y cuántos) estudiantes utilizan correo *hotmail* pues simplemente introducimos la siguiente sentencia SQL

```
SELECT email FROM students WHERE email ~ 'hotmail.com';
```

Observen el nuevo comando

"WHERE"->"donde (satisfazga la condición)"

y el uso de la virgulilla como comparador lógico indicando que dicha condición tenga la cadena de texto 'hotmail.com'. Otros detalles a considerar es el uso de las comillas simples y el uso de mayúsculas: si usamos comillas dobles se interpreta como nombre de columna y si usamos mayúsculas NO devuelve los correos electrónicos ya que "hoTMail.com" NO ES IGUAL A "hotmail.com":

"hoTMail.com" "hotmail.com"

Sobre nomenclatura del correo electrónico:

Esto nos plantea desde ya nuevos retos: si vamos a desarrollar una aplicación seria es deber que los usuarios ingresen direcciones de correo válidos (sintaxis *usuario@dominio*):

- El nombre del *usuario* NO debe llevar espacios a menos que estén precedidos por una barra inversa y entrecomillado.
- El *dominio* NUNCA debe llevar espacios (ni arrobas, claro está).
- No pueden haber espacios antes o después de una dirección válida.
- No pueden llevar dobles puntos "..", así estén entrecomillados.
- El nombre del *usuario* no pueden contener dobles arrobas a menos que estén debidamente entrecomillados.

Y son sólo algunas de las reglas que aplican a la sintaxis de una dirección de correo electrónico, para mayor información (en inglés, ¡cuando no!) en RFC 822 y dirección de correo electrónico . La buena noticia es que PosgreSQL admite varios lenguajes de programación que junto con *triggers* o "disparadores" nos permitirán capturar y revisar los correos electrónicos antes de agregarlos a la base de datos (serán tratados a futuro en una entrada aparte). Y aunque nos hemos salido un poco de las consultas sencillas era necesario ir abonando el terreno hacia temas más avanzados (oh idioma castellano, qué poético eres, nunca cambies ;-)).

Os ruego hagan la prueba escribiendo las sentencias en sus múltiples variantes, como ayuda os dejo el comando **LIKE** cuyo uso tiene una sintaxis más elaborada:

```
SELECT email FROM students WHERE email LIKE '%hotmail.com%';
```

Produciendo el mismo resultado que con la virgulilla; vale destacar que al colocar el símbolo de porcentaje como comodín le estamos ordenando que nos busque cualquier dirección de correo electrónico registrado en nuestra tabla *students* que contenga la cadena de caracteres "hotmail.com" (obsérvese que si algún bromista registrara la dirección de correo electrónico "hotmail.com@gmail.com" sería devuelta también en la consulta, os animo a probarla -aunque ~~dudo que alguien en verdad~~ tenga ésa dirección-) pero eso aún no resuelve nuestro problema de mayúsculas y minúsculas.

Una solución sería utilizar una *función* integrada por defecto en nuestra base de datos, es decir, una serie de programas ya registrados y de propósito general; dicha función para este caso es **LOWER()** aplicado al campo *email* (ojo que si es así lo que entrecomillamos como condición **debe ir todo en minúsculas**):

```
SELECT email FROM students WHERE LOWER(email) LIKE '%hotmail.com%';
```

o también podemos usar, como al principio, la virgulilla que nos ahorra el uso de comodines:

```
SELECT email FROM students WHERE LOWER(email) ~ 'hotmail.com';
```

Si quereís ver el resto de funciones de cadena, haced click aquí.

Por último, si quiséramos el caso contrario, **listar los estudiantes que no tienen correo "hotmail" registrado:**

```
SELECT email FROM students WHERE LOWER(email) !~ 'hotmail.com';
```

y lo único que hicimos fue agregarle el operador lógico "!" que indica negación "NOT", el cual si que lo podemos usar con LIKE:

```
SELECT email FROM students WHERE LOWER(email) NOT LIKE '%hotmail.com%';
```

Y entonces nos desconectamos de la base de datos como es debido:

.