Practica D

Comandos Específicos de PostgreSQL

1. Objetivo General

Conocer comandos específicos de PostgreSQL relativos a seguridad, búsqueda de texto, condicionales y manipulación de cadenas.

2. Objetivos Secundarios

- Conocer comandos de seguridad que ofrece PostgreSQL.
- Conocer comandos de búsqueda de texto en PostgreSQL.
- Conocer comandos para la evaluación de condiciones en PostgreSQL.
- Profundizar en la manipulación de cadenas en PostgreSQL.

3. Introducción

A lo largo de este manual se han presentado características generales, funcionalidades y elementos comunes que resultan útiles para una buena parte de los individuos que tienen contacto con bases de datos. Sin embargo, los SMBDs poseen muchas más características que permiten realizar funciones muy particulares, para los fines de esta práctica se presentarán algunos de ellos, mismos que tienen que ver con seguridad, manipulación de cadenas y búsquedas sobre texto.

La seguridad en un SMBD es un factor muy importante hoy en día, el enfrentarse a diferentes amenazas en la red resulta común, es por ello que el conocer formas de proteger los datos y restringirlos a ciertos tipos de usuarios resulta conveniente.

Por otra parte, los SMBDs ofrecen funciones para realizar búsquedas de cadenas de caracteres sobre atributos que almacenan una gran cantidad de texto. Esto resulta útil a la hora de buscar cadenas específicas que se encuentran entre esa vasta información.

Finalmente manipular cadenas, ya sea al momento de introducir datos o al operar con elementos ya almacenados en la base de datos, es una tarea que permite adaptar la forma en que serán presentados al usuario. Realizar esta manipulación desde el SMBD evita tener que recurrir a un lenguaje de propósito general en consecuencia dicha labor se vuelve más eficiente.

En la siguiente sección se presentarán la sintaxis y uso de comandos específicos para manipular cadenas, realizar búsquedas sobre texto y mejorar la seguridad.

Como anexo en la parte final de la práctica, se presenta un comparativo entre los comandos de SQL Server y PostgreSQL, mostrando diferencias y similitudes.

4. Desarrollo

4.1 Encriptar en PostgreSQL 9.3.4.

En PostgreSQL existen diferentes formas de encriptar y desencriptar datos, en esta sección se muestra la manera de realizarlo, dando la sintaxis y uso de los mismos.

SHA1[1]

Este algoritmo debe crearse primero, esto se debe a que PostgreSQL no configura por default la función SHA1 para encriptar. Este comando se usa dentro una instrucción INSERT.

Sintaxis:

```
SHA1('dato_a_encriptar')
```

Argumentos:

- `SHA1`
  Algoritmo reservado en PostgreSQL.
- `(`
  Paréntesis abierto, indica que se iniciaran las instrucciones para encriptar el dato.
- `'dato_a_encriptar'`
  Es el dato a encriptar y que es de tipo nvarchar, char, varchar, binary, varbinary o nchar. Las comillas simples son necesarias.
- `)`
  Paréntesis que cierra indica el final de la instrucción del algoritmo SHA1.

Uso:

Para hacer uso del algoritmo SHA1, es necesario el crearlo antes, para posteriormente ser usado, lo anterior se realiza con el siguiente código, ver Figura D.54.

[1] http://www.PostgreSQLql.org/docs/current/interactive/pgcrypto.html

```
CREATE OR REPLACE FUNCTION sha1(bytea) RETURNS TEXT AS $$
SELECT encode(digest($1, 'sha1'), 'hex')
$$ LANGUAGE SQL strict immutable;
```

Figura D.54 Creación del algoritmo SHA1.

Dónde[2]:

- `CREATE OR REPLACE FUNCTION`
  Crea o reemplaza una función en PostgreSQL.
- `sha1(bytea)`
  Función llamada 'sha1' que recibe como argumento una cadena de datos binarios[3].
- `RETURNS TEXT`
  Parte de CREATE OR REPLACE FUNCTION que regresa un texto después de evaluar sha1(bytea).
- `AS $$`
  Indica que la función creada 'sha1' es diferente a la función que existe en PostgreSQL.
- `digest($1, 'sha1')`
  Función hash que recibe una cadena de datos binaria y el algoritmo a usar, en este caso 'sha1', esta función hash devuelve una cadena de datos binaria.
- `encode(digest($1,'sha1'),'hex')`
  Codifica el resultado de digest en hexadecimal[4].
- `SELECT`
  Selecciona el dato codificado en hexadecimal.
- `LANGUAGE SQL strict`
  Indica el lenguaje de la función, donde 'SQL strict ' es definido por el usuario o algún lenguaje compatible con PostgreSQL,
- `inmutable`
  Indica que la función no puede ser modificada en la base de datos y siempre regresa el mismo resultado evaluado con los mimos valores en los argumentos.
- `;`
  Símbolo 'punto y coma' indica el fin de la creación de la función 'sha1'.

Una vez creado el algoritmo se crea la tabla 'Encriptar' y se inserta un dato encriptado con 'SHA1', ver Figura D.55 y Figura D.56.

[2] http://www.PostgreSQLql.org/docs/9.3/static/sql-createfunction.html
[3] http://www.PostgreSQLql.org/docs/9.3/static/datatype-binary.html
[4] http://www.PostgreSQLql.org/docs/9.3/static/functions-string.html

```
CREATE TABLE Encriptar(Nombre VARCHAR(200), Contraseña TEXT);
```

Figura D.55 Creación de tabla Encriptar.

```
INSERT INTO Encriptar(Nombre, Contraseña) VALUES ('Alberto', sha1('micontraseña'));
```

Figura D.56 Inserción del dato encriptado.

Seleccionando los datos se observa el dato encriptado en la tabla, ver Figura D.57 y Figura D.58.

```
SELECT * FROM Encriptar;
```

Figura D.57 Selección del dato encriptado.

	nombre character varying(200)	contraseña text
1	Alberto	4c95d933ca952553330724b809dd61344aad5b6b

Figura D.58 Vista del dato encriptado.

Para comprobar la contraseña, se compara el password dentro de una clausula WHERE, ver Figura D.59 y Figura D.60.

```
SELECT Nombre FROM Encriptar WHERE Contraseña=sha1('micontraseña');
```

Figura D.59 Comprobación del dato encriptado.

	nombre character varying(200)
1	Alberto

Figura D.60 Vista del dato encriptado.

MD5[5]

Este algoritmo se usa dentro una de una consulta SELECT como en una instrucción INSERT.

Sintaxis:

```
MD5('dato_a_encriptar')
```

Argumentos:

- MD5
  Algoritmo reservado en PostgreSQL.
- (

[5] http://www.PostgreSQLql.org/docs/9.3/static/functions-string.html

Paréntesis abierto, indica que se iniciaran las instrucciones para encriptar el dato.

- 'dato_a_encriptar'
  Es el dato a encriptar, éste debe ser de tipo nvarchar, char, varchar, binary, varbinary o nchar. Las comillas simples son necesarias.
-)
  Paréntesis que cierra indica el final de la instrucción del algoritmo MD5.

Uso:

Se inserta un dato encriptado con 'MD5' en la tabla 'Encriptar', antes creada, ver Figura D.55 y Figura D.61.

```
INSERT INTO Encriptar(Nombre, Contraseña) VALUES ('Alberto', MD5('micontraseña'));
```

Figura D.61 Inserción del dato encriptado.

Seleccionando los datos se observa el dato encriptado en la tabla, ver Figura D.62 y Figura D.63.

```
SELECT * FROM Encriptar;
```

Figura D.62 Selección del dato encriptado.

	nombre character varying(200)	contraseña text
1	Alberto	4c95d933ca952553330724b809dd61344aad5b6b
2	Alberto	2deae7fb1fb94ee8e23919b093b26982

Figura D.63 Vista del dato encriptado.

Para comprobar la contraseña, se compara el password dentro de una clausula WHERE, ver Figura D.64 y Figura D.65.

```
SELECT Nombre FROM Encriptar WHERE Contraseña=MD5('micontraseña');
```

Figura D.64 Comprobación del dato encriptado.

	nombre character varying(200)
1	Alberto

Figura D.65 Vista del dato encriptado.

4.2 Búsqueda de texto en PostgreSQL versión 9.3.4.

Dentro de la búsqueda de texto se pueden encontrar diferentes comandos. A continuación se presentan diferentes formas y usos de dichos comandos, así como la sintaxis de cada uno.

LIKE[6]

El comando LIKE funciona para buscar palabras en las columnas de la tabla, tiene diferentes combinaciones para calcular la búsqueda.

Sintaxis:

LIKE 'cadena_a_buscar'
LIKE 'cadena_a_buscar%'
LIKE '%cadena_a_buscar%'
LIKE '_cadena_a_buscar_'

Argumentos:

- LIKE Palabra reservada en Postgres para hacer la búsqueda de texto en las columnas.
- cadena_a_buscar Es la cadena que será buscada en la tabla, definida por el usuario.
- % Símbolo 'porcentaje', si esta al inicio de cadena_a_buscar, busca la cadena a buscar sin importar lo que este a su izquierda. Si % está a la derecha busca coincidencias sin importar lo que este después de la derecha.
- _ Símbolo 'guion bajo' indica que solo buscara coincidencias a su derecha o izquierda con diferencia de un carácter.

Uso:

El uso de LIKE se muestra a continuación, ver Figura D.66 a D.73.

```
SELECT * FROM "Salarios" WHERE Nombre LIKE 'a'
```

Figura D.66 Selección de consulta de Salarios sin restricción.

[6] http://www.tutorialspoint.com/postgresql/postgresql_like_clause.htm

id integer	nombre character varying(255)	apellido character varying(255)	salario integer	totalsalario integer

Figura D.67 Vista de consulta de Salarios sin restricción.

```
SELECT * FROM "Salarios" WHERE Nombre LIKE 'a%'
```

Figura D.68 Selección de Salarios con restricción.

id integer	nombre character varying(255)	apellido character varying(255)	salario integer	totalsalario integer

Figura D.69 Vista de Salarios con restricción.

```
SELECT * FROM "Salarios" WHERE Nombre LIKE '%a%'
```

Figura D.70 Selección de Salarios con restricción.

id integer	nombre character varying(255)	apellido character varying(255)	salario integer	totalsalario integer
1	Joan	Head	56	12
3	Chase	Stein	61	26
4	Desirae	Owens	47	37
6	Sonya	Maldonado	46	9
8	Dai	Valdez	17	3
10	Giacomo	Juarez	98	28
11	Jermaine	Strong	48	71
14	Elijah	Vaughan	64	66
15	Raja	Roy	88	33
19	Paul	Woods	90	44
21	Darryl	Stephens	91	35
22	Graham	Kirk	76	35
23	Alexander	Joyner	59	45
25	Ryan	Chavez	20	1
28	Gabriel	Hayden	16	67
30	Leandra	Cote	73	98
31	Nadine	Reilly	24	44
32	Ignacia	Barr	77	90
33	Palmer	Stone	79	79

Figura D.71 Vista de Salarios con restricción.

```
SELECT * FROM "Salarios" WHERE Nombre LIKE '_a_'
```

Figura D.72 Selección de Salarios con restricción.

id integer	nombre character varying(255)	apellido character varying(255)	salario integer	totalsalario integer
8	Dai	Valdez	17	3

Figura D.73 Vista de Salarios con restricción.

4.3 Manipulación de fechas en PostgreSQL versión 9.3.4.

AGE[7]

Dentro del comando AGE se tienen dos versiones, una para calcular el tiempo que ha transcurrido de una fecha a otra, y en la segunda versión se calcula el tiempo transcurrido desde una fecha en particular a la fecha del sistema, ambas versiones de esta instrucción se usan dentro de un SELECT.

Sintaxis:

```
Versión1:
AGE(TIMESTAMP 'fecha_mas_actual', TIMESTAMP 'fecha_antigua')

Versión 2:
AGE(TIMESTAMP 'fecha_antigua')
```

Argumentos:

- AGE
  Palabra reservada en PostgreSQL.
- (
  Paréntesis que abre, indica el inicio de los argumentos para calcular el tiempo transcurrido.
- TIMESTAMP
  Palabra reservada en PostgreSQL.
- 'fecha_mas_actual'
  Indica la fecha hasta donde se calculara el tiempo transcurrido, las comillas simples son necesarias, el formato de fecha será: Año, Mes y Día (aaaa/mm/dd) indicando la separación con un '-' o '/'.
- ,
  Símbolo 'coma' indica la separación entre argumentos de fecha.

[7] http://www.PostgreSQLql.org/docs/9.3/static/functions-datetime.html

- 'fecha_antigua'
  Indica la fecha desde donde se calculara el tiempo transcurrido.
-)
  Paréntesis que cierra, indica el fin de la instrucción AGE.

Uso:

El uso de AGE presenta la siguiente estructura para las versiones antes mencionadas, ver Figuras D.74 a D.77.

```
SELECT AGE(TIMESTAMP '2003/04/15', TIMESTAMP '1997/06/13') AS TIEMPO_TRANSCURRIDO;
```

Figura D.74 Consulta del tiempo transcurrido.

tiempo_transcurrido interval
5 years 10 mons 2 days

Figura D.75 Vista del tiempo transcurrido.

```
SELECT AGE(TIMESTAMP '1957/06/13') AS TIEMPO_TRANSCURRIDO;
```

Figura D.76 Consulta del tiempo transcurrido a la fecha del sistema.

tiempo_transcurrido interval
56 years 11 mons 2 days

Figura D.77 Vista del tiempo transcurrido a la fecha del sistema.

NOW[8]

Esta instrucción devuelve la fecha y hora del sistema, se usa dentro una instrucción SELECT.

Sintaxis:

NOW ( )

Argumentos:

- NOW
  Palabra reservada en PostgreSQL.
- (
  Paréntesis que abre, indica el inicio de los argumentos para la fecha y hora, en este caso no hay argumentos.
-)

[8] http://www.PostgreSQLql.org/docs/9.3/static/functions-datetime.html

Paréntesis que cierra, indica el final de la instrucción NOW.

Uso:

Se muestra el uso básico del comando NOW, ver Figura D.78 y Figura D.79.

```
SELECT NOW() AS FECHA_HORA;
```

Figura D.78 Selección de FECHA y HORA del sistema.

fecha_hora timestamp with time zone
2014-05-15 08:58:44.597-05

Figura D.79 Vista de FECHA y HORA del sistema.

CURRENT_DATE

Devuelve la fecha actual del sistema, se usa dentro de una instrucción SELECT.

Sintaxis:

```
CURRENT_DATE
```

Argumentos:

- `CURRENT_DATE`
  Palabra reservada en PostgreSQL.

Uso:

El comando CURRENT_DATE se usa en PostgreSQL como sigue, ver Figura D.80 y Figura D.81.

```
SELECT CURRENT_DATE AS FECHA;
```

Figura D.80 Selección de FECHA.

fecha date
2014-05-15

Figura D.81 Vista de FECHA.

CURRENT_TIME

Devuelve el tiempo del sistema, fecha y hora, se usa dentro de una instrucción SELECT.

Sintaxis:

```
CURRENT_TIME
```

Argumentos:

- `CURRENT_TIME`
  Palabra reservada en PostgreSQL.

Uso:

El comando CURRENT_TIME es usado en PostgreSQL como sigue, ver Figura D.82 y Figura D.83.

```
SELECT CURRENT_TIME AS HORA;
```

Figura D.82 Selección de HORA.

hora time with time zone
09:28:54.293-05

Figura D.83 Vista de HORA.

4.4 Condicionales en PostgreSQL versión 9.3.4.

CASE-WHEN[9]

Así como en el caso de SQL Server, en PostgreSQL CASE-WHEN evalúa una lista de condiciones y devuelve una de las expresiones de resultado posibles, se usa en cualquier instrucción o clausula, por ejemplo instrucciones como SELECT, UPDATE, DELETE y SET, y en clausulas como SELECT_LIST, IN, WHERE, ORDER BY y HAVING.

Existen dos tipos de CASE-WHEN, un formato simple y la segunda de formato de búsqueda.

Sintaxis:

CASE sencillo:

```
CASE inicio_expresion
WHEN expresion_comparada THEN resultado_expresion
ELSE resultado_en_otro_caso
```

[9] http://www.PostgreSQLql.org/docs/9.3/static/functions-conditional.html

CASE de búsqueda:

```
CASE
WHEN expresion_booleana THEN resultado_expresion
ELSE resultado_en_otro_caso
```

Argumentos:

- `CASE`
  Palabra reservada en PostgreSQL.
- `inicio_expresion`
  Es una expresión evaluada en caso de usarse CASE sencillo.
- `WHEN`
  Palabra reservada en PostgreSQL.
- `expresion_comparada`
  Es una expresión sencilla valida que se compara con inicio_expresion cuando se usa CASE sencillo, tanto expresion_comparada como inicio_expresion deben ser iguales o deben ser una conversión implícita.
- `THEN`
  Palabra reservada en PostgreSQL.
- `resultado_expresion`
  Esta es la expresión que se devuelve cuando inicio_expresion y expresion_comparada son iguales.
- `ELSE`
  Palabra reservada en PostgreSQL y puede ser opcional.
- `resultado_en_otro_caso`
  Devuelve la expresión si ninguna comparación es igual. Si se omite este argumento y ninguna comparación es verdadera, CASE devuelve NULL.
- `expresion_booleana`
  Expresión booleana que se evalúa cuando se utiliza CASE de búsqueda y es cualquier expresión booleana valida.

Uso:

Para el uso del comando CASE-WHEN se crea una tabla de Salarios, insertando 100 tuplas en ella, ver Figura D.84.

```sql
DROP TABLE IF EXISTS "Salarios";

CREATE TABLE "Salarios" (
  id SERIAL PRIMARY KEY,
  Nombre varchar(255) default NULL,
  Apellido varchar(255) default NULL,
  Salario integer NULL,
  TotalSalario integer NULL
);
```

Figura D.84 Creación de tabla Salarios.

El uso de CASE-WHEN sencilla presenta la siguiente estructura, ver Figura D.85. Así como los resultados después de consultar la tabla y las condiciones, ver Figura D.86.

```sql
--Expresión CASE sencilla
SELECT Nombre, Apellido, TotalSalario,
       CASE TotalSalario
            WHEN 80 THEN 'Aprobado'
            WHEN 50 THEN 'No Aprobado'
            ELSE 'Sin revision'
            END
FROM "Salarios";
```

Figura D.85 Selección de Salarios sencilla.

	nombre character varying(255)	apellido character varying(255)	totalsalario integer	case text
1	Joan	Head	80	Aprobado
2	Kirsten	Camacho	50	No Aprobado
3	Chase	Stein	50	No Aprobado
4	Desirae	Owens	80	Aprobado
5	Riley	Mcfadden	47	Sin revision
6	Sonya	Maldonado	9	Sin revision
7	Gregory	Solomon	43	Sin revision
8	Dai	Valdez	50	No Aprobado
9	Amery	Hickman	14	Sin revision
10	Giacomo	Juarez	80	Aprobado

Figura D.86 Vista de tuplas de Salarios sencilla.

El uso de CASE-WHEN de búsqueda presenta la siguiente estructura, ver Figura D.87. Así como los resultados después de consultar la tabla y las condiciones, ver Figura D.88.

```sql
--Expresion CASE de busqueda
SELECT Nombre, Apellido, TotalSalario,
       CASE
            WHEN TotalSalario > '80' THEN 'Aprobado'
            WHEN TotalSalario < '50' THEN 'No Aprobado'
            ELSE 'Sin revision'
            END
FROM "Salarios";
```

Figura D.87 Selección de salarios con búsqueda.

	nombre character varying(255)	apellido character varying(255)	totalsalario integer	case text
1	Joan	Head	12	No Aprobado
2	Kirsten	Camacho	79	Sin revision
3	Chase	Stein	26	No Aprobado
4	Desirae	Owens	37	No Aprobado
5	Riley	Mcfadden	47	No Aprobado
6	Sonya	Maldonado	9	No Aprobado
7	Gregory	Solomon	43	No Aprobado
8	Dai	Valdez	3	No Aprobado
9	Amery	Hickman	14	No Aprobado

Figura D.88 Vista de tuplas de Salarios con búsqueda.

COALESCE[10]

PostgreSQL ofrece el comando COALESCE tomando los argumentos por orden y regresa el primer valor que no es nulo dentro de cada expresión.

Sintaxis:

```
COALESCE( expresion1, expresion2,..., expresionN )
```

Argumentos:

- `COALESCE`
  Palabra reservada en PostgreSQL.
- `(`
  Paréntesis que abre, indica el inicio de los argumentos a evaluar por COALESCE.
- `expresionX`
  Representa una expresión de cualquier tipo.
- `,`
  El símbolo 'coma' indica la separación entre las expresiones a evaluarse por COALESCE.
- `)`
  Paréntesis que cierra, indica el término de los argumentos a evaluar por COALESCE.

Uso:

Usando la tabla 'Salarios', ver Figura D.84, se hace uso del comando COALESCE, así como la vista del resultado, ver Figura D.89 y Figura D.90.

[10] http://www.PostgreSQLql.org/docs/9.3/static/functions-conditional.html

```
SELECT Nombre, Apellido,
       COALESCE(Salario, TotalSalario) AS PRIMER_VALOR_NO_NULO
FROM "Salarios";
```

Figura D.89 Selección de tuplas con comando COALESCE.

	nombre character varying(255)	apellido character varying(255)	primer_valor_no_nulo integer
1	Joan	Head	56
2	Kirsten	Camacho	55
3	Chase	Stein	61
4	Desirae	Owens	47
5	Riley	Mcfadden	64
6	Sonya	Maldonado	46
7	Gregory	Solomon	69
8	Dai	Valdez	17
9	Amery	Hickman	15

Figura D.90 Vista de tuplas con comando COALESCE.

NULLIF[11]

Este comando al igual que en SQL Server, en PostgreSQL regresa un valor NULL si las expresiones que se comparan son iguales.

Sintaxis:

```
NULLIF(expresion, expresion)
```

Argumentos:

- `NULLIF`
  Palabra reservada en PostgreSQL.
- `(`
  Paréntesis que abre, indica el inicio de las expresiones a comparar por NULLIF.
- `expresion`
  Representa una expresión de cualquier tipo.
- `,`
  El símbolo 'coma' indica la separación entre las expresiones a evaluarse por NULLIF.
- `)`
  Paréntesis que cierra, indica el término de las expresiones a evaluar por NULLIF.

Uso:

[11] http://www.PostgreSQLql.org/docs/9.3/static/functions-conditional.html

El uso de NULLIF se describe a continuación, usando la tabla "Salarios" creada anteriormente, ver Figura D.84, así como la vista del resultado, ver Figura D.91 y Figura D.92.

```
SELECT Nombre, Apellido,
       NULLIF(Salario, TotalSalario) AS NULOS
FROM "Salarios";
```

Figura D.91 Selección de datos de Salarios con NULLIF.

	nombre character varying(255)	apellido character varying(255)	nulos integer
1	Joan	Head	56
2	Kirsten	Camacho	55
3	Chase	Stein	61
4	Desirae	Owens	47
5	Riley	Mcfadden	64
6	Sonya	Maldonado	46
7	Gregory	Solomon	69
8	Dai	Valdez	17
9	Amery	Hickman	15

Figura D.92 Vista de datos Salarios con NULLIF.

GREATEST y LEAST[12]

GREATEST regresa el valor más grande entre la lista de argumentos, LEAST regresa el valor más pequeño entre la lista de argumentos. Se usa dentro de un SELECT.

Sintaxis:

```
GREATEST(expresion1, expresion2,..,expresionN)
 LEAST(expresion1, expresion2,..,expresionN)
```

Argumentos:

- GREATEST
  Palabra reservada en PostgreSQL.
- LEAST
  Palabra reservada en PostgreSQL.
- (
  Paréntesis que abre, indica el inicio de los argumentos a evaluar por GREATEST o LEAST.
- expresionX
  Esta expresión es de tipo numérico, como puede ser columnas de tipo numérico o expresiones que se evalúen a un dato numérico. Si la expresión es NULL, tanto

[12] http://www.PostgreSQLql.org/docs/9.3/static/functions-conditional.html

GREATEST y LEAST ignora la expresión, regresara NULL si todas las expresiones son NULL.

- ,
  Símbolo ‘coma’ indica la separación de las expresiones a evaluar.
-)
  Paréntesis que cierra, indica el fin del comando GREATEST o LEAST.

Uso:

El uso de GREATEST como LEAST se presenta de la siguiente manera, ver Figuras D.93 a D.96, tomando como base la tabla Salarios, ver Figura D.84.

```
SELECT GREATEST(Salario, TotalSalario) AS MAX_COLUMNAS FROM "Salarios";
```

Figura D.93 Selección de valores máximos de Salarios con GREATEST.

	max_columnas integer
1	56
2	79
3	61
4	47
5	64
6	46
7	69
8	17
9	15
10	98

Figura D.94 Vista de valores máximos de Salarios con GREATEST.

```
SELECT LEAST(Salario, TotalSalario) AS MAX_COLUMNAS FROM "Salarios";
```

Figura D.95 Selección de valores mínimos de Salarios con LEAST.

	max_columnas integer
1	12
2	55
3	26
4	37
5	47
6	9
7	43
8	3
9	14
10	28

Figura D.96 Vista de valores mínimos de Salarios con LEAST.

4.5 Manipulación de cadenas en PostgreSQL versión 9.3.4.

Dentro de la manipulación de cadenas podemos encontrar comandos como:

II y CONCAT[13]

Devuelve una cadena que es resultado de unir dos o más valores de cadena.

Sintaxis:

```
CONCAT( 'valor_cadena1', 'valor_cadena2',..,'valor_cadenaN')
```

Argumentos:

- `CONCAT`
  Palabra reservada en PostgreSQL para concatenar cadenas.
- `(`
  Paréntesis que abre, indica el inicio de los valores de las cadenas a concatenar.
- `'valor_cadenaX'`
  Indica un valor de cadena que se va a concatenar con los demás valores, las comillas simples son necesarias.
- `,`
  Símbolo 'coma' indica la separación de los valores de las cadenas.
- `)`
  Paréntesis que cierra, indica el final de la instrucción de CONCAT.

Uso:

El uso de CONCAT presenta la siguiente estructura, ver Figura D.97 y Figura D.98.

```
SELECT CONCAT('HOLA ', 'ESTO ES ', 'UNA ', 'CADENA ', 'CONCATENADA') AS CADENA_CONATENADA;
```

Figura D.97 Concatenar cadena.

	cadena_conatenada text
1	HOLA ESTO ES UNA CADENA CONCATENADA

Figura D.98 Vista de Concatenar cadena.

CHARACTER_LENGTH o CHAR_LENGTH [14]

Tanto CHARACTER_LENGTH como CHAR_LENGTH regresa el número de caracteres en la cadena especificada incluyendo espacios en blanco finales.

[13] http://www.PostgreSQLql.org/docs/9.3/static/functions-string.html
[14] http://www.PostgreSQLql.org/docs/9.3/static/functions-string.html

Sintaxis:

```
CHARACTER_LENGTH('expresion_cadena')
O
CHAR_LENGTH('expresion_cadena')
```

Argumentos:

- CHARACTER_LENGTH y CHAR_LENGTH
  Palabras reservadas en PostgreSQL.
- (
  Paréntesis que abre, indica el inicio de la expresión de la cadena a contar.
- 'expresion_cadena'
  Expresión de caracteres a contar, las comillas simples son necesarias.
-)
  Paréntesis que cierra, indica el final de la instrucción LEN.

Uso:

El uso de CHARACTER_LENGTH y CHAR_LENGTH se muestran a continuación, ver Figuras D.99 a D.102.

```
SELECT CHARACTER_LENGTH('Total de caracteres en la cadena ') AS Total_CAracteres;
```

Figura D.99 Uso de CHARACTER_LENGTH en una cadena.

	total_caracteres integer
1	33

Figura D.100 Vista de CHARACTER_LENGTH en la cadena.

```
SELECT CHAR_LENGTH('Total de caracteres en la cadena ') AS Total_Caracteres;
```

Figura D.101 Uso de CHAR_LENGTH en una cadena.

	total_caracteres integer
1	33

Figura D.102 Vista de CHAR_LENGTH en la cadena.

OVERLAY[15]

Este comando reemplaza parte de una cadena por caracteres.

[15] http://www.PostgreSQLql.org/docs/9.3/static/functions-string.html

Sintaxis:

```
OVERLAY('cadena_a_sustituir' PLACING 'subcadena_sustituida'
        FROM inicio_sustitucion FOR fin_sustitucion)
```

Argumentos:

- `CONCAT`
  Palabra reservada en PostgreSQL.
- `(`
  Paréntesis que abre, indica el inicio de los argumentos para la sustitución de la cadena.
- `'cadena_a_sustituir'`
  Es una sucesión de caracteres, esta puede contener espacios, las comillas simples son necesarias.
- `PLACING`
  Palabra reservada en PostgreSQL.
- `'subcadena_sustituida'`
  Es una cadena de caracteres a sustituir dentro de 'cadena_a_sustituir', puede contener espacios, las comillas simples son necesarias.
- `FROM`
  Palabra reservada en PostgreSQL.
- `inicio_sustitucion`
  Indica la posición en 'cadena_a_sustituir' donde se empezara a sustituir 'subcadena_sustituida', es de tipo INT.
- `FOR`
  Palabra reservada en PostgreSQL, puede ser opcional a partir de este comando.
- `fin_sustitucion`
  Indica la posición final hasta donde se sustituirá la subcadena en la cadena, es de tipo INT. Si no se indica FOR con fin_sustitucion, 'subcadena_sustituida' tomara solo su longitud y se sustituirá en 'cadena_a_sustituir'.

Uso:

Su uso se representa como sigue, ver Figura D.103 y Figura D.104.

```
SELECT OVERLAY('Cxxxxx y sustituir' PLACING 'adena a' FROM 2 FOR 7);
```

Figura D.103 Sustitución de cadena con OVERLAY.

	overlay text
1	Cadena a sustituir

Figura D.104 Vista de cadena con OVERLAY.

LOWER[16]

Convierte en minúsculas los caracteres en mayúsculas, devolviendo la expresión de caracteres.

Sintaxis:

```
LOWER(expresion_caracteres)
```

Argumentos:

- `LOWER`
  Palabra reservada en PostgreSQL.
- `(`
  Paréntesis que abre, indica el inicio de la expresión a convertir en minúsculas.
- `expresion_caracteres`
  Es una expresión de datos binarios o de caracteres, puede ser una constante, una variable o una columna, esta expresión deber ser de tipo que se pueda convertir explícitamente a VARCHAR.
- `)`
  Paréntesis que cierra, indica el fin de la instrucción LOWER.

Uso:

LOWER presenta la siguiente estructura, usando además el comando CONCAT, ver Figura D.105 y Figura D.106.

```
SELECT LOWER(
       CONCAT('HOLA ', 'ESTO ES ', 'UNA ', 'CADENA ', 'CONCATENADA ', 'EN MINUSCULAS '))
       AS CADENA_CONATENADA_EN_minusculas;
```

Figura D.105 Selección de la cadena concatenada convertida en minúsculas.

	cadena_conatenada_en_minusculas text
1	hola esto es una cadena concatenada en minusculas

Figura D.106 Vista de la cadena concatenada convertida en minúsculas.

UPPER[17]

Convierte en mayúsculas los caracteres en minúsculas, devolviendo la expresión de caracteres.

[16] http://www.PostgreSQLql.org/docs/9.3/static/functions-string.html
[17] http://www.PostgreSQLql.org/docs/9.3/static/functions-string.html

Sintaxis:

```
UPPER(expresion_caracteres)
```

Argumentos:

- `UPPER`
  Palabra reservada en PostgreSQL.
- `(`
  Paréntesis que abre, indica el inicio de la expresión a convertir en mayúsculas.
- `expresion_caracteres`
  Es una expresión de datos binarios o de caracteres, puede ser una constante, una variable o una columna, esta expresión deber ser de tipo que se pueda convertir explícitamente a VARCHAR.
- `)`
  Paréntesis que cierra, indica el fin de la instrucción UPPER.

Uso:

UPPER presenta la siguiente estructura, usando además el comando CONCAT, ver Figura D.107 y Figura D.108.

```
SELECT UPPER(
       CONCAT('hola ', 'esto es ', 'una ', 'cadena ', 'concatenada ', 'en MAYUSCULAS '))
           AS CADENA_CONATENADA_EN_minusculas;
```

Figura D.107 Selección de la cadena concatenada convertida en MAYUSCULAS.

	cadena_conatenada_en_minusculas text
1	HOLA ESTO ES UNA CADENA CONCATENADA EN MAYUSCULAS

Figura D.108 Vista de la cadena concatenada convertida en MAYUSCULAS.

POSITION[18]

Devuelve la posición inicial de una cadena buscada en otra si esta se encuentra.

Sintaxis:

```
POSITION('expresion_a_encontrar' IN 'expresion_donde_buscar')
```

[18] http://www.PostgreSQLql.org/docs/9.3/static/functions-string.html

Argumentos:

- `POSITION`
  Palabra reservada en PostgreSQL.
- `(`
  Paréntesis que abre, indica el inicio de los argumentos para buscar el índice en la cadena.
- `'expresion_a_encontrar'`
  Es una expresión de caracteres que contiene la secuencia que se va a buscar.
- `IN`
  Palabra reservada en PostgreSQL.
- `'expresion_donde_buscar'`
  Es la expresión donde se buscara 'expresion_a_encontrar'.
- `)`
  Paréntesis que cierra, indica el final de la instrucción POSITION

Uso:

Haciendo uso de la instrucción SELECT se usa POSITION, siguiendo una combinación con los comandos UPPER y CONCAT, la cual se representa como sigue, ver Figura D.109 y Figura D.110.

```
SELECT POSITION('CADENA' IN (SELECT  UPPER(
       CONCAT('hola ', 'esto es ', 'una ', 'cadena ', 'concatenada ', 'en MAYUSCULAS '))
           AS CADENA_CONATENADA_EN_MAYUSCULAS)) AS POSICION_DE_CADENA;
```

Figura D.109 Comando POSITION combinado con UPPER y CONCAT.

	posicion_de_cadena integer
1	18

Figura D.110 Vista del comando POSITION posición 'CADENA'.

SUBSTRING[19]

SUBSTRIGN devuelve parte de una expresión de caracteres, se describe el comando básico.

Sintaxis:

```
SUBSTRING( expresion FROM inicio FOR longitud)
```

[19] http://www.PostgreSQLql.org/docs/9.3/static/functions-string.html

Argumentos:

- SUBSTRING
  Palabra reservada en PostgreSQL.
- (
  Paréntesis que abre, indica el inicio de los argumentos para la sub-cadena a obtener.
- expresion
  Es una expresión de caracteres donde se buscara la sub-cadena de longitud indicada.
- FROM
  Palabra reservada en PostgreSQL.
- inicio
  Es de tipo entero que específica donde comienzan los caracteres devueltos. Si inicio es menor que 1, la expresión devuelta comenzara en el primer carácter especificado en <expresion>.
- FOR
  Palabra reservada en PostgreSQL.
- longitud
  Es un estero positivo que especifica cuantos caracteres de 'expresion' se van a devolver. Si la longitud es negativa, se genera un error y finaliza la instrucción. Si la suma de 'inicio' y 'longitud' es mayor que el número de caracteres de 'expresion', se devuelve la expresion de valor completa que empieza en 'inicio'.
-)
  Paréntesis que cierra, indica el fin de la instrucción SUBSTRING.

Uso:

El siguiente ejemplo muestra la estructura de uso de SUBSTRING combinado con UPPER y CONCAT, ver Figura D.111 y Figura D.112.

```
SELECT SUBSTRING(
       (SELECT
           UPPER(
       CONCAT('hola ', 'esto es ', 'una ', 'cadena ', 'concatenada ', 'en MAYUSCULAS '))) FROM 6 FOR 15)
           AS SUBCADENA;
```

Figura D.111 Selección de sub-cadena de longitud 15.

	subcadena text
1	ESTO ES UNA CAD

Figura D.112 Vista de sub-cadena de longitud 15.

SUBSTR[20]

Devuelve parte de una expresión de caracteres, binaria de texto o de imagen.

Sintaxis:

```
SUBSTR( expresion, inicio, longitud)
```

Argumentos:

- `SUBSTR`
  Palabra reservada en PostgreSQL.
- `(`
  Paréntesis que abre, indica el inicio de los argumentos para la sub-cadena a obtener.
- `expresion`
  Es una expresión de caracteres donde se buscara la sub-cadena de longitud indicada.
- `,`
  Símbolo 'coma' indica la separación de los argumentos de la cadena a sustraer.
- `inicio`
  Es de tipo entero que específica donde comienzan los caracteres devueltos. Si inicio es menor que 1, la expresión devuelta comenzara en el primer carácter especificado en <expresion>.
- `longitud`
  Es un estero positivo que especifica cuantos caracteres de 'expresion' se van a devolver. Si la longitud es negativa, se genera un error y finaliza la instrucción. Si la suma de 'inicio' y 'longitud' es mayor que el número de caracteres de 'expresion', se devuelve la expresión de valor completa que empieza en 'inicio'.
- `)`
  Paréntesis que cierra, indica el fin de la instrucción SUBSTR.

Uso:

El siguiente ejemplo muestra la estructura de uso de SUBSTRING combinado con UPPER y CONCAT, ver Figura D.113 y Figura D.114.

[20] http://www.PostgreSQLql.org/docs/9.3/static/functions-string.html

```
SELECT SUBSTR(
       (SELECT
           UPPER(
       CONCAT('hola ', 'esto es ', 'una ', 'cadena ', 'concatenada ', 'en MAYUSCULAS '))), 6, 15)
           AS SUBCADENA;
```

Figura D.113 Selección de sub-cadena de longitud 15.

	subcadena text
1	ESTO ES UNA CAD

Figura D.114 Vista de sub-cadena de longitud 15.

LTRIM, RTRIM, TRIM, BTRIM[21]

El comando LTRIM devuelve una expresión de caracteres después de quitar el patrón de caracteres a la izquierda en la expresión. RTRIM devuelve una expresión de caracteres después de quitar el patrón de caracteres a la derecha en la expresión. A diferencia de los comandos antes mencionados, TRIM quita todos los patrones de caracteres que encuentre en la expresión, tanto a la izquierda como la derecha. BTRIM quita todos los patrones de carácter que encuentre en la expresión, no importando el orden en que se encuentren.

Sintaxis:

```
LTRIM('expresion_cadena', 'patron_caracteres')
RTRIM('expresion_cadena', 'patron_caracteres')
 TRIM('expresion_cadena', 'patron_caracteres')
BTRIM('expresion_cadena', 'patrón_caracteres')
```

Argumentos:

- `LTRIM, RTRIM, TRIM y BTRIM`
  Palabras reservadas PostgreSQL.
- `(`
  Paréntesis que abre, indica el inicio de la expresión a modificar.
- `'expresion_cadena'`
  Es una cadena de caracteres en la cual se harán las modificaciones dependiendo el comando.
- `,`
  Símbolo 'coma' indica la separación de los argumentos.
- `'patron_caracteres'`
  Es una cadena de carácter(es) que indica la eliminación de los mismos en 'expresion_cadena'.
- `)`

[21] http://www.PostgreSQLql.org/docs/9.3/static/functions-string.html

Paréntesis que cierra, indica el final de la instrucción de LTRIM o RTRIM.

Uso:

LTRIM se representa de la siguiente manera, ver Figura D.115 y Figura D.116.

```
SELECT LTRIM('xxxyy Conjunto de palabras sin patron de caracteres a la izquierda', 'xy')
       AS Comando_LTRIM;
```

Figura D.115 Selección de cadena con comando LTRIM.

	comando_ltrim text
1	Conjunto de palabras sin patron de caracteres a a la izquierda

Figura D.116 Vista de cadena con comando LTRIM.

RTRIM se representa de la siguiente manera, ver Figura D.117 y Figura D.118.

```
SELECT RTRIM('Conjunto de palabras sin patron de caracteres a la derecha: xxxxyy', 'xy')
       AS Comando_RTRIM;
```

Figura D.117 Selección de cadena con comando RTRIM.

	comando_rtrim text
1	Conjunto de palabras sin patron de caracteres a la derecha:

Figura D.118 Vista de cadena con comando RTRIM.

TRIM se representa como sigue, ver Figura D.119 y Figura D.120.

```
SELECT TRIM('xxxxyyy Conjunto de palabras sin patron de caracteres a la izquierda ni derecha xxxxyy', 'xy')
       AS Comando_TRIM;
```

Figura D.119 Selección de cadena con comando TRIM.

	comando_trim text
1	Conjunto de palabras sin patron de caracteres a la izquierda ni derecha

Figura D.120 Vista de cadena con comando TRIM.

Y por último se presenta BTRIM como sigue, ver Figura D.121 y Figura D.122.

```
SELECT BTRIM('xxxyyyxxyyx Conjunto de palabras sin patron de caracteres yxxxx', 'xy')
       AS Comando_BTRIM;
```

Figura D.121 Selección de cadena con comando BTRIM.

	comando_btrim text
1	Conjunto de palabras sin patron de caracteres

Figura D.122 Vista de cadena con comando BTRIM.

REPLACE[22]

Este comando reemplaza todas las instancias de un patrón de cadena por otra cadena en la expresión inicial.

Sintaxis:

```
REPLACE('expresion_cadena', 'patrón_cadena',
        'cadena_reemplazar')
```

Argumentos:

- `REPLACE`
  Palabra reservada en PostgreSQL.
- `(`
  Paréntesis que abre, indica el inicio de argumentos para reemplazar en la cadena.
- `'expresion_cadena'`
  Es la cadena donde se va a buscar el patrón indicado, esta expresión puede ser de tipo binario o de caracteres, las comillas simples son necesarias.
- `,`
  Símbolo 'coma' indica la separación de los argumentos en la instrucción REPLACE.
- `'patron_cadena'`
  Es una subcadena a buscar dentro de expresion_cadena y puede ser de tipo binario o de caracteres, pero no debe ser vacía (''), las comillas simples son necesarias.
- `'cadena_reemplazar'`
  Esta es la cadena que se reemplaza por el patron_cadena y puede ser de tipo binario o de caracteres, las comillas simples son necesarias.
- `)`
  Paréntesis que cierra, indica el final de la instrucción REPLACE.

Uso:

El siguiente ejemplo muestra el uso del comando REPLACE, sustituyendo 'a' por 'i', ver Figura D.123 y Figura D.124.

[22] http://www.PostgreSQLql.org/docs/9.3/static/functions-string.html

```
SELECT REPLACE('cadena a reemplazar por', 'a', 'i') AS REEMPLAZO_CADENA;
```

Figura D.123 Reemplaza 'a' por 'i' en la cadena.

	reemplazo_cadena text
1	cideni i reemplizir por

Figura D.124 Vista de la cadena reemplazada.

5. Ejercicios

a) Ejecuta cada uno de los comandos aquí presentados en alguna de las tablas de tu base de datos poblada[23].
b) Investiga otros comandos para encriptar y desencriptar datos, realizar búsquedas de texto y manipulación de cadenas.

Entregables requeridos para prácticas subsecuentes:

- Agrega un atributo encriptado a alguna de las tablas de tu base de datos.
- Agrega un atributo en el que se registre la fecha de inserción de manera automática.

Anexo A

Tabla 1. Comparativa de comandos entre SQL Server 2012 y PostgreSQL 9.3.4.

SQL Server 2012	PostgreSQL 9.3.4	Descripción	Diferencias
ENCRYPTBYCERT	SHA1	Encripta datos.	
DECRYPTBYCERT	MD5	Desencripta datos.	
CONTAINS	No existe	Búsqueda de datos	
FREETEXT	No existe	Búsqueda de datos	
GETDATE	NOW	Devuelve la fecha y hora del sistema.	Ninguna.
DATEDIFF	No existe	Calcula límites entre valores.	
No existe	AGE	Calcula el tiempo transcurrido a una fecha.	
DATE	CURRENT_DATE	Devuelve la fecha del sistema.	Se usa CURRENT_TIMESTAMP en SQL Server 2012 para

[23] Ver Practica C

			hacer la equivalencia en PostgreSQL 9.3.4.
TIME	CURRENT_TIME	Devuelve la hora del sistema.	Se usa CURRENT_TIMESTAMP en SQL Server 2012 para hacer la equivalencia en PostgreSQL 9.3.4.
CASE-WHEN	CASE-WHEN	Evalúa expresiones y devuelve otros resultados en las expresiones.	Difieren en la forma de expresar la selección de las expresiones.
COALESCE	COALESCE	Devuelve los valores no nulos dentro de cada expresión.	Ninguna.
NULLIF	NULLIF	Devuelve un valor NULL si las expresiones son iguales.	Ninguna.
MAX y MIN	GREATEST y LEAST	Devuelve el valor máximo y mínimo de la expresión	MAX y MIN, solo evalúa un solo argumento. GRASTEST y LEAST evalúa más de un argumento.
CONCAT	CONCAT	Une dos o más valores de tipo cadena.	Ninguna.
LEN	CHARACTER_LENGTH O CHAR_LENGTH	Regresa el número de caracteres en la expresión.	Ninguna.
LOWER	LOWER	Convierte en minúsculas la cadena expresada.	Ninguna.
UPPER	UPPER	Convierte en mayúsculas la cadena expresada.	Ninguna.
CHARINDEX	POSITION	Regresa la posición inicial de una cadena dentro de otra.	Ninguna.
SUBSTRING	SUBSTRING	Regresa parte de una expresión de caracteres.	Difiere la sintaxis una de otra.
No existe	SUBSTR	Devuelve parte de una expresión de caracteres, su sintaxis en diferente que SUBSTRING.	

LTRIM y RTRIM		Quita espacios a la izquierda y derecha, respectivamente.	
	LTRIM y RTRIM	Quita un patrón de caracteres a la izquierda o derecha, respectivamente de la expresión.	
No existe	TRIM	Quita todos los patrones de caracteres tanto a la izquierda como a la derecha en una expresión.	
No existe	BTRIM	Quita todos los patrones de caracteres tanto a la izquierda como a la derecha en una expresión.	
REPLACE	REPLACE	Reemplaza un patrón de cadena en otra.	Ninguna.
No existe	OVERLAY	Reemplaza parte de una cadena por caracteres.	