Universidad Nacional Autónoma de México
Facultad de Ciencias

Fundamentos de Bases de Datos

Semestre 2022-2

Práctica 10: Procedimientos y disparadores

Profesor:
Gerardo Áviles Rosas

Ayudantes de Teoría:
Gerardo Uriel Soto Miranda
Rocío Aylin Huerta González

Ayudantes de Laboratorio:
Ricardo Badillo Macías Rodrigo
Alejandro Sánchez Morales

Mayo 25, 2022

Funciones

- Una función que reciba la CURP y nos regrese la edad del cliente:

  `getEdadCliente(curp TEXT)`

  Como el CURP tiene la fecha de nacimiento completa a partir del quinto caracter, entonces optamos por manipular el string (`TEXT`) que recibimos como parámetro.

  Primero lo descomponemos en años (índices 5-6), meses (7-8) y dias (9-10). Luego convertimos a números (`integer`) y al tener CURPs generados aleatoreamente operamos de la siguiente manera:

  - Si los años son mayores a 21, entonces sumamos 1900, en otro caso sumamos 2000. Ésto con el objetivo de no tener años que aún no hayan transcurrido.
  - Como únicamente existen meses del 1 al 12, entonces si el número sale de éste rango lo regresamos a 12 (decidido arbitrariamente).
  - Finalmente, para no tener que considerar los años bisisestos y las variaciones de días dependiendo del mes, si el mes es 2 y los días mayores a 28, entonces los truncamos en 28. Además todos los días mayores a 30 se regresan a 15 (decidido arbitrariamente).

  Ya teniendo el CURP procesado, concatenemos años, meses y días en un string para poder sacar la diferencia en años entre la fecha generada y el momento en el que se llame la función.

- Una función que reciba idEstetica y regrese las ganancias de esa estetica.

  Tenemos dos funciones diferente para éste caso:

  1. `gananciasEstetica(idEstetica TEXT)`

     Al introducir el nombre de la Estética (idEstetica), hacemos uso de la función `SUM()` para obtener los ingresos totales de la estética en cuestión registrados en la tabla Recibo. Posteriormente agrupamos éste resultado con el nombre de la estética correspondiente y regresamos.

     Lo anterior fue lo que se implementó inmediatamente tras leer las especificaciones de la función solicitada. Sin embargo, no pudimos dejar de pensar en que éstas no son las ganancias totales de la estética, pues aún faltaría descontar los gastos hechos (en este caso salarios). Entonces llegamos a la siguiente función.

  2. `gananciasEsteticaNeta(idEstetica TEXT)`

     Ésta función une las tablas de Veterinario, Estilista y Supervisor con la de Estética y sus ingresos para poder descontar los salarios de los trabajadores y así obtener las ganancias (o pérdidas en algunos casos) de la estética.

     Lo anterior se consigue al unir Veterinario con VeterinarioTrabajaEn y agrupar por Estética, para después sumar los salarios de los veterinarios (por Estética) y evitar filas repetidas al momento de unir con las demás tablas.

     Después hacemos exactamente lo mismo para las tablas Estilista y EstilistaTrrabajaEn.

     Ahora, como las estéticas solo tienen un supervisor no hace falta operar de manera similar a Veterinario y Estilista, por lo que simplemente unimos con las dos tablas obtenidas.

     Finalmente, obtenemos el total de salarios a pagar y lo restamos a los ingresos de la Estética. Con ésto obtenemos las ganancias de la Estética.

Observación. Ésta función nos regresa nombre de la estética, pagos, ingresos totales y ganancias al llamarla de la siguiente manera:

```
SELECT salarios(idEstetica);
```

Por lo que para obtener únicamente las ganancias:

```
SELECT ganancias FROM salarios(idEstetica);
```

Procedimientos

- Un SP el cual se encarga de registrar un veterinario, en este SP, debes introducir la información del veterinario y se debe encargar de insertar en las tablas correspondientes.

  `veterinarioInsert(...)`

  Lo primero de lo que tuvimos que ocuparnos fue ver que no tuvieramos Veterinarios no asigandos a alguna Estética, por lo que además de los datos del veterinario, también se requiere el nombre de alguna Estética.

  Ahora, toca revisar cada uno de los campos pasados a la función:

  - La Estética tiene que existir.
  - El CURP debe tener exactamente 18 caracteres.
  - El nombre completo no puede tener números y/o simbolos, tampoco puede ser vacío (’␣’).
  - El número en la dirección debe componerse únicamente de digitos.
  - El código postal deben ser sólo números y, al ser un string, longitud exactamente igual a 5.
  - La calle no puede ser vacía (’␣’).
  - El número telefónico debe tener únicamente digitos y longitud igual a 10.
  - El salario no puede ser negativo.
  - El RFC debe tener exactamente 13 caracteres.
  - No puede tener un número de pacientes negativo.

  Si alguno de los campos no cumple con las restricciones, entonces lanzamos una excepción acorde.

  Una vez que verificamos todos los datos introducidos realizamos un `INSERT`, primero en Veterinario y luego en VeterinarioTrabajaEn.

- Un SP el cual se encarga de eliminar un veterinario, en este SP debes introducir el CURP del Veterinario y cuando elimines este veterinario se debe eliminar todas sus referencias.

  `veterinarioDelete(idCurp TEXT)`

  Tras obtener el CURP del veterinario que se quiere eliminar del registro, realizamos las siguientes vefificaciones:

  1. Que el CURP tenga exactamente 18 caracteres.
  2. Y que el CURP exista en la tabla Veterinario.

  Luego de cumplir con las restricciones anteriores realizamos `DELETE`, primero en VeterinarioTrabajaEn y después en Veterinario.

Trigers

- Un trigger que se encargue de invertir el apellido paterno con el apellido materno de los supervisores.

  `invertir`

  Como tenemos que invertir los apellidos cada vez que se insertara o actualizara la tabla Supervisor. Primero establecimos la función `swap()`, la cuál será la encargada de reasignar los valores para `apellidoPaterno` y `apellidoMaterno`.

  Luego creamos el trigger para que actúe antes de hacer `INSERT` o `UPDATE`.

- Un trigger que se encargue de evitar que se pueda modificar y borrar de la tabla mascota.

  `sinModificaciones`

  Al pedirnos que el trigger impida la modificación de la tabla (supusimos `UPDATE` y `DELETE`), entonces se nos ocurrió que simplemente podríamos lanzar una excepción antes de que se lleve a cabo alguna de las dos operaciones para así interrumpirla.

  Entonces tenemos la función `stopChanges()` que se encarga de levantar las excepciones en `UPDATE` o `DELETE`.

  Finalmente creamos el trigger para que se dispare antes de que se realice alguna de las operaciones.