Trabajo Práctico 2. Programación Lógica

Puntuación

Puntaje Total: 100 puntos
Aprobación: 60 puntos
Fecha de entrega: 18/11/2019 23:55 Hs.

Condiciones de entrega

1. El presente trabajo práctico deberá resolverse en grupo de hasta 3 (tres) personas.
2. Entrega: Se realizará por medio del campus virtual de la UTN, en la tarea correspondiente al TP 2. La extensión del archivo será `.zip` o `.rar`, de acuerdo al programa de compresión usado. El nombre del archivo se consigue concatenando un prefijo del número del TP con los apellidos de los integrantes separados por guiones (Ej: Pérez y Abdala, el nombre será `tp2-abdala-perez.zip`). Note que no hay espacios en blanco ni acentos en el nombre de archivo. Dentro del archivo de entrega, deben constar los siguientes:
   - Fuentes SWI-Prolog: Se debe entregar un archivo denominado `tp2.pl`
   - Los casos de prueba se entregarán en un archivo de texto, no deben ser capturas de pantalla. Deberán cubrir diferentes resultados que puedan obtenerse de la evaluación de los predicados solicitadas. Se enfatiza que se adjunten casos de prueba que sean claros, válidos y suficientes para poder probar el trabajo.
     Nombre del archivo: `casos-de-prueba.txt`.
   - Archivo de texto (`integrantes.txt`) con una línea por cada integrante en la cual figure el nombre del alumno/a y su dirección de email.
3. Penalización por entrega fuera de término: Si el trabajo práctico se entrega después de la fecha indicada, y hasta una semana tarde, tendrá una quita de 15 puntos. No serán recibidos trabajos luego de una semana de la fecha de entrega. Los trabajos que se deban rehacer/corregir fuera de la fecha de entrega tienen una quita de 30 puntos.

Descripción del problema

Muchas aplicaciones utilizan bases de datos. El código de la aplicación requiere que los datos estén arreglados de cierta manera denominada esquema de datos. Es muy popular utilizar sistemas de gestión de datos relacionales, basados en el

lenguaje SQL para realizar las consultas. Un sistema de gestión de bases de datos permite crear bases de datos. Una base de datos contiene datos cargados en un esquema determinado. Por ejemplo, una base de datos de una aplicación comercial puede tener en su esquema una tabla para almacenar datos de clientes, otra para almacenar el catálogo de productos de su oferta comercial, otra para guardar las ventas realizadas, etc. La tabla de clientes está definida por las columnas que la componen, por ejemplo: nombre y apellido, CUIT, domicilio de entrega de productos, etc. Las columnas contienen datos uniformes, por ejemplo, el CUIT es una cadena de caracteres, la fecha de venta es una fecha, y así sucesivamente. Nos referimos como **tipo** al conjunto de valores posibles para una columna dada (esta no es la terminología SQL pero ayudará en el TP).

Uno de los problemas que se presentan al crear una aplicación que utiliza bases de datos es conseguir los datos iniciales para cargar en las tablas recién creadas, y por lo tanto vacías. Algunos de estos datos provienen de fuentes externas y son constantes o de poca variación, como la lista de provincias de nuestro país, la lista de países del mundo, la lista de códigos postales argentinos, los departamentos que conforman la organización, etc. Otros datos se crearán mientras la aplicación funciona y son propios de la utilización de la propia aplicación; ejemplos son: los clientes (con sus nombres, CUIT, etc.), las ventas (con sus fechas de venta, productos vendidos, etc.), y otros.

Un listado de los datos fijos se puede conseguir de una vez al inicio del desarrollo, y cargar en forma de lote (carga *batch*) en la base de datos. Éste es un procedimiento bien conocido.

Sin embargo, durante el desarrollo de una aplicación que nunca ha sido implementada conseguir datos de clientes y ventas no deja de ser un problema interesante. Se deben “inventar” datos para que la aplicación puede utilizarse. Imaginen una aplicación de venta de electrodomésticos que no tiene ningún producto para poner en el carrito de compras, o peor aún, no tiene ninguna sucursal.

Hay otro motivo para utilizar datos inventados. Las empresas deben cumplir regulaciones de privacidad de datos y acuerdos de confidencialidad con sus clientes y proveedores. Entonces aún cuando la aplicación esté en producción y tengamos datos reales a disposición, no podemos usar esos datos para que los desarrolladores hagan sus pruebas, por el riesgo de la exposición que representa.

Nuestro trabajo será escribir un programa en Prolog que genere datos ficticios de manera pseudoaleatoria para diferentes columnas de las tablas de una aplicación.

Implementación

Se solicita modelar el dominio del problema utilizando cláusulas lógicas, definiendo en PROLOG los predicados que se describen a continuación.

Parte 1

En la parte 1 de la solución escribiremos los predicados que generan valores elegidos de manera pseudoaleatoria para los dominios. La lista es la siguiente, incluyendo ejemplos de resultados para la variable de la consulta:

Predicado	Resultado ejemplo
`fake_nombre(-Nombre)` Un `Nombre` es un *string* compuesto por entre uno y tres nombres de pila y uno o dos apellidos. Los nombres de pila y apellidos se deben obtener de manera independiente. Debe concatenar los *strings* componentes para obtener el `Nombre`.	`"Juan Perez"` `"Ariel Ezequiel Gomez"`

`fake_direccion(-Direccion)`
Una `Dirección` se compone de calle, número de casa, piso y departamento. El piso y el departamento serán opcionales en algunas de las direcciones generadas. Los componentes indicados se deben obtener de los predicados que siguen.
	`"Francia 1347 Piso 1 Dto B"`
	`"San Martin 1243"`
	`"Urquiza 5443 Piso 5"`
	`"Corrientes 3421 Dto H"`

`fake_calle(-Calle)`	`"San Martin"`
`fake_numero_casa(-Numero)` Los `Numero` de casa son números naturales entre 1000 y 15000.	`2348`
`fake_piso(-Piso)` El `Piso` es un número natural entre 1 y 40, o puede ser también el símbolo `pb`.	`3`

`fake_numero_departamento(-Dpto)` El número de `Dpto` es un número natural entre 1 y 20, o es una letra en el rango desde A hasta H.	`8`
`fake_ciudad(-Ciudad)`	`"Santa Fe"`
`fake_pais(-Pais)`	`"Peru"`
`fake_provincia(-Provincia)`	`"Cordoba"`
`fake_nombre_color(-NombreColor)`	`rojo`
`fake_nombre_empresa(-NombreEmpresa)`	`"El Sur Programadores"`
`fake_moneda(-Sigla, -Nombre).`	`usd, "Dolar estadounidense"`
`fake_fecha(-Fecha, +Desde, +Hasta).` Una `Fecha` es una lista de tres números, que corresponden respectivamente al año, mes y día. La fecha resultante debe ser posterior o igual a `Desde` y anterior o igual a `Hasta`.	`[2019, 12, 23]`
`fake_dia_de_semana(-DiaDeSemana).`	`jueves`

Parte 2

En la parte 2 del trabajo práctico vamos a construir una tabla de datos a partir de la descripción de sus columnas:

```
genera_tabla_datos(+EspecificacionTabla, -Tabla)
```

La especificación de la tabla es una función:

```
especificacion_tabla(CantidadRenglones, ListaColumnas)
```

donde la lista de columnas es una lista de la funcion `col/2`, en la cual el primer argumento es el nombre de la columna, y el segundo es el nombre del tipo de dicha columna.

Por ejemplo, una descripción de columna puede ser:

```
col(id, autoincremento(100))
```

que significa que el nombre de la columna en la tabla es `id`, y el tipo es `autoincremento` con valores que comienzan en `100`.

Los nombres de tipo posibles son los términos siguientes:

- `autoincremento(Desde)`
- `nombre`
- `direccion`
- `calle`
- `numero_casa`
- `piso`
- `numero_departamento`
- `ciudad`
- `pais`
- `provincia`
- `nombre_color`
- `nombre_empresa`
- `moneda_sigla`
- `moneda_nombre`
- `fecha(Desde, Hasta)`
- `dia_de_semana`

El tipo `autoincremento/1` indica que en el primer renglón este valor se corresponde con `Desde`, y cada renglón siguiente tiene como valor uno más que el anterior generado. En una tabla dada, sólo puede haber una columna designada con `autoincremento/1`.

El tipo `fecha(Desde, Hasta)` indica que se debe generar una fecha en ese rango de fechas.

A los fines del TP, los casos de datos fijos (`ciudad`, `pais`, `provincia`, `nombre_color`, `moneda_sigla`, `moneda_nombre`, `dia_de_semana`) deben contener al menos 5 valores diferentes entre sus opciones.

Ejemplo de consulta

```
genera_tabla_datos( especificacion_tabla(10,
[ col(id, autoincremento(100)),
  col(nombre, nombre),
  col(fec_nac, fecha([1920, 1, 1], [2001, 12, 31]) ]), Tabla )
```

El resultado en `Tabla` es una función `tabla(Cabecera, Datos)`, donde `Cabecera` es una lista de los nombres de columnas de la tabla, y `Datos` es una lista de renglones o registros; siendo cada renglón una lista con los datos. Por ejemplo:

```
Tabla = tabla([id, nombre, fecha_nac], [
  [100, "Juan Perez", [1942, 8, 24]],
  [101, "Daniela Gomez", [1983, 2, 18]],
…
])
```

Los registros son 10, cuyos `id` van del 100 al 109.

Referencias útiles (SWI-Prolog)

- **Generación de números pseudoaleatorios:** Para obtener un número pseudoaleatorio X en el rango 0 a 10, con distribución uniforme, utilizar:

  ```
  X is random(10).
  ```

  Ver documentación en:
  https://www.swi-prolog.org/pldoc/doc_for?object=f(random/1)

  Para generar siempre la misma secuencia pseudoaleatoria, usar:
  https://www.swi-prolog.org/pldoc/man?predicate=set_random/1

- **Predicados disponibles para manipular *strings*:**

  Ver documentación en:
  https://www.swi-prolog.org/pldoc/man?section=string-predicates