Segundo Trabajo de Bases de Datos 2 (20%)

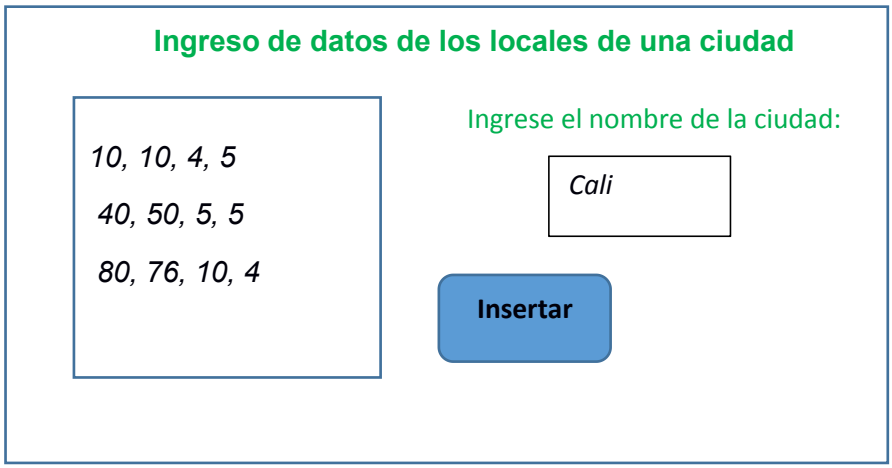
Por favor lea todo el enunciado y los ejemplos.

Cada punto vale 25%

Primer punto.

Supongamos que en cada ciudad los locales (por ejemplo, de tiendas) se representan mediante rectángulos. Cada rectángulo está definido por cuatro números enteros tal y como lo requiere el método *drawRect* de Java: *drawRect(int x, int y, int width, int height)*.

Haga un formulario en Java a través del cual el usuario inserta los datos de los locales de una ciudad, así:



En el ejemplo anterior, el usuario ingresa los datos de 3 locales (rectángulos) de Cali. Los datos de cada rectángulo son 4 números enteros separados por comas: x, y, width, height.

Nota: Se garantiza que los datos se ingresarán de esta forma y que no tienen errores. Se garantiza que todos los rectángulos de una ciudad están ubicados en un plano de 500 x 500 (es decir, cada ciudad es un cuadrado de 500 x 500, delimitada por las coordenadas (0, 0) y (499, 499)). Se garantiza además que cada rectángulo (local) tiene área (es decir, no se trata de una línea recta ni de un punto).

Por supuesto, cada ciudad puede tener muchos locales, en este ejemplo Cali solo tiene 3.

Cuando el usuario presione el botón Insertar, en la base de datos Oracle se deben insertar los datos en una columna de tipo XMLTYPE. El documento XML **debe** quedar así:

```
<locales>
 <rectangulo>
   <a>10</a>
   <b>10</b>
   <c>4</c>
```

```
  <d>5</d>
 </rectangulo>
 <rectangulo>
  <a>40</a>
  <b>50</b>
  <c>5</c>
  <d>5</d>
 </rectangulo>
 <rectangulo>
  <a>80</a>
  <b>76</b>
  <c>10</c>
  <d>4</d>
 </rectangulo>
</locales>
```

Es decir, en la base de datos debe quedar una tabla llamada CITY con una columna Nombre_ciudad (**clave primaria**) y la otra llamada Locales con el XML de sus locales:

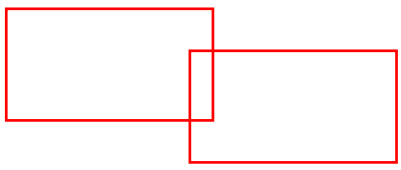
Tabla: CITY

Nombre_ciudad	Locales
Cali	XML de locales
Medellín	XML de locales
Bogotá	XML de locales

Como se observa, habrá varias ciudades cada una con su XML de locales...

Se debe validar que en cada ciudad los rectángulos **no se solapen**, es decir, **entre los rectángulos de una ciudad no debe haber intersección**; sin embargo, **sí** pueden tener bordes o esquinas en común. Esta validación la puede hacer directamente en Java o si prefiere puede hacerla mediante un *trigger* en la base de datos.

Así, los siguientes locales de una ciudad son inválidos:



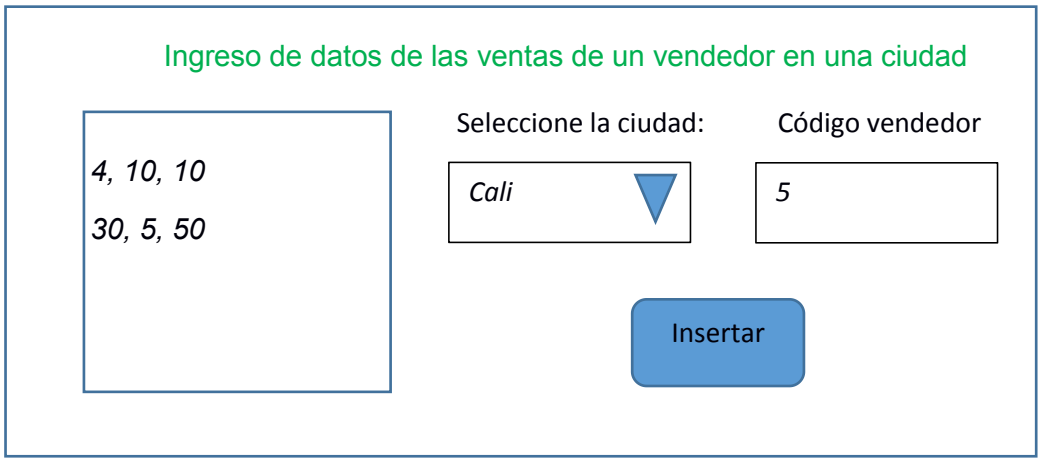
Segundo punto.

En cada ciudad hay vendedores que deambulan por ella. Interesa guardar los datos de las ventas de cada vendedor. Los datos de **una** venta de un vendedor en una ciudad son:

x, y, v

Donde x, y son las coordenadas donde el vendedor hizo la venta y v el valor de la venta.

Haga un formulario en Java a través del cual el usuario inserta los datos de las ventas de un vendedor en una ciudad, así:



En el ejemplo anterior, se insertan 2 ventas del vendedor con código 5 en Cali (note que la ciudad se selecciona de una lista de selección, donde estarán todas las ciudades que hasta el momento se han insertado en la tabla CITY por medio del primer formulario). Estas ventas fueron así: Una tuvo lugar en las coordenadas (4, 10) por un valor de $10 y la otra por un valor de $50 en las coordenadas (30, 5).

Nota: Se garantiza que los datos se ingresarán de esta forma y que no tienen errores. Se garantiza además que todos las coordenadas x, y de todas las ventas están dentro de un plano de 500 x 500 y que v es un número de máximo dos digitos.

Por supuesto, un vendedor puede tener muchas ventas. Los datos de cada venta son 3 números enteros separados por comas: x, y, v.

Cuando el usuario presione el botón Insertar, en la base de datos Oracle se deben insertar los datos en una tabla llamada VVCITY así:

Tabla: VVCITY

CodigoVendedor	Ciudad	Ventas
5	Cali	4, 10, 10 30, 5, 50

La clave primaria de esta tabla es compuesta: {CodigoVendedor, Ciudad}.

La columna Ventas **debe ser una tabla anidada**, donde en cada una de sus filas se guarda una venta.

Se debe controlar lo siguiente: si al momento de insertar las ventas de un vendedor en una ciudad, varias de sus ventas ocurrieron en el **mismo** punto, entonces se debe insertar un solo punto con la suma de sus ventas. Ejemplo:

Si el usuario ingresa en el formulario estas ventas para un vendedor con código 88 en Cali:

4, 10, 10

8, 1, 20

4, 10, 5

4, 10, 10

La tabla anidada del vendedor 88 en Cali debe quedar así:

4, 10, 25
8, 1, 20

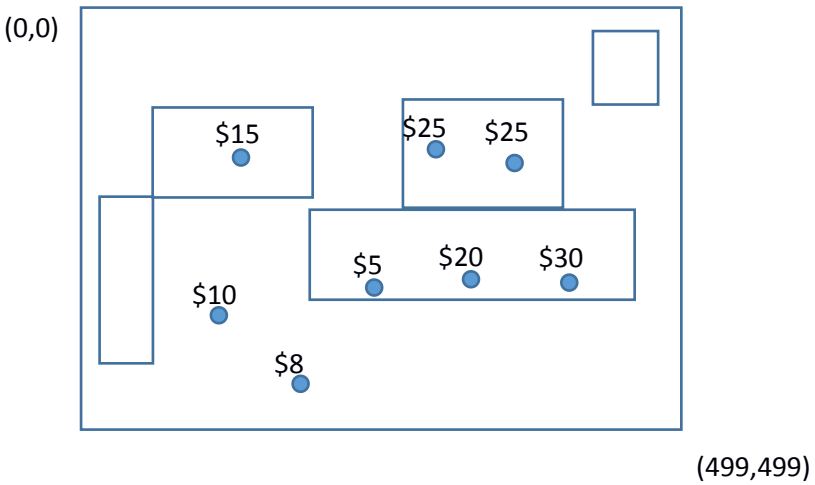
Este control lo puede hacer en Java o mediante un trigger.

Tercer punto.

Haga un formulario en Java a través del cual el usuario selecciona una ciudad (mediante una lista de selección) y el formulario le grafique todos los locales de la ciudad y todos los puntos donde ha habido ventas (junto al punto se debe mostrar el valor de la venta).

Veamos un ejemplo.

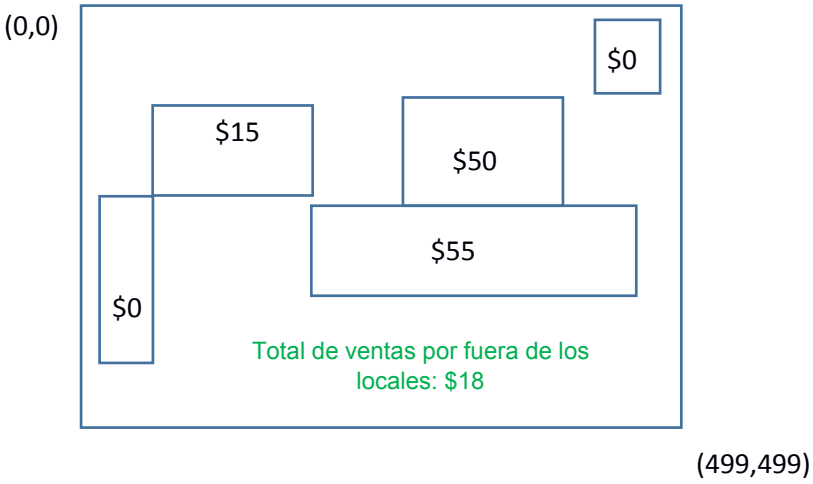
Supongamos que el usuario selecciona Bogotá, entonces se debe mostrar algo como lo siguiente:



Nota: si en una ciudad en un mismo punto (x, y) ocurren varias ventas, se debe mostrar el valor de la **suma** de todas las ventas que ocurrrieron en dicho punto. Por ejemplo, si en Cali ocurrieron estas dos ventas: (4, 10, 10) y (4, 10, 25); entonces al graficar el punto (4, 10) debe salir con un valor de $35.

Cuarto punto.

Haga un formulario en Java a través del cual el usuario selecciona una ciudad (mediante una lista de selección) y el formulario le grafique todos los locales de la ciudad e indica cual fue el total de ventas dentro de cada local. Se debe mostrar también el total de ventas que no ocurrieron dentro de los locales. Considerando el ejemplo mostrado en el tercer punto, la salida para Bogotá debe ser:



Nota: si una venta ocurre en el **borde (o precisamente en una esquina)** de un local, usted decide si considera que dicha venta está adentro o afuera del local.

Notas adicionales:

- Sus soluciones deben funcionar para cualquier cantidad de filas que tengan las tablas. Los datos presentados son solo para ejemplificar.
- No cambie los nombres indicados para las tablas (ni para las columnas), ni les adicione ni les quite columnas**.**
- Puede usar todas las estructuras de datos y todas las tablas auxiliares que desee.
- Puede hacer un formulario principal que llame a todos los demás o puede hacer formularios independientes para cada punto.
- Para facilitar la visualización, puede modificar la escala (multiplicando por algún factor).
- Para entregar por email a *fjmoreno@unal.edu.co*, el martes 6 de octubre **hasta las 5 pm**. Solo se califican trabajos enviados a ese correo.
- **No se reciben trabajos en hora posterior**. No se reciben versiones "mejoradas". No se califican trabajos enviados "por accidente" a otros correos.
- **Se debe incluir un informe donde se describa cómo se solucionó cada punto.** Este informe hace parte de la calificación del trabajo (máximo 4 hojas). **No enviar los datos de prueba que usted usó para probar sus códigos.**

- Grupos de **tres** personas.
- Los trabajos deben ser independientes entre los grupos. Trabajos copiados **así sea en un SOLO punto** se califican con 0 (cero) en su totalidad para todos los integrantes. Las soluciones presentadas deben ser originales. El trabajo debe ser desarrollado por los integrantes del grupo no por personas ajenas a él.
- El monitor les puede ayudar con aspectos técnicos pero su función **no** es hacerles la práctica **ni está autorizado** para **cambiarle las condiciones del trabajo**.
- Recuerde que este trabajo evalúa la parte de Java y Oracle. **Si trabaja con otras herramientas**, así su trabajo funcione y sea "espectacular", el trabajo **NO** será calificado.
- **No** se califica la "belleza" de los formularios, se califica la funcionalidad.
- Cualquier duda consultarla con el profesor.

Francisco Moreno