

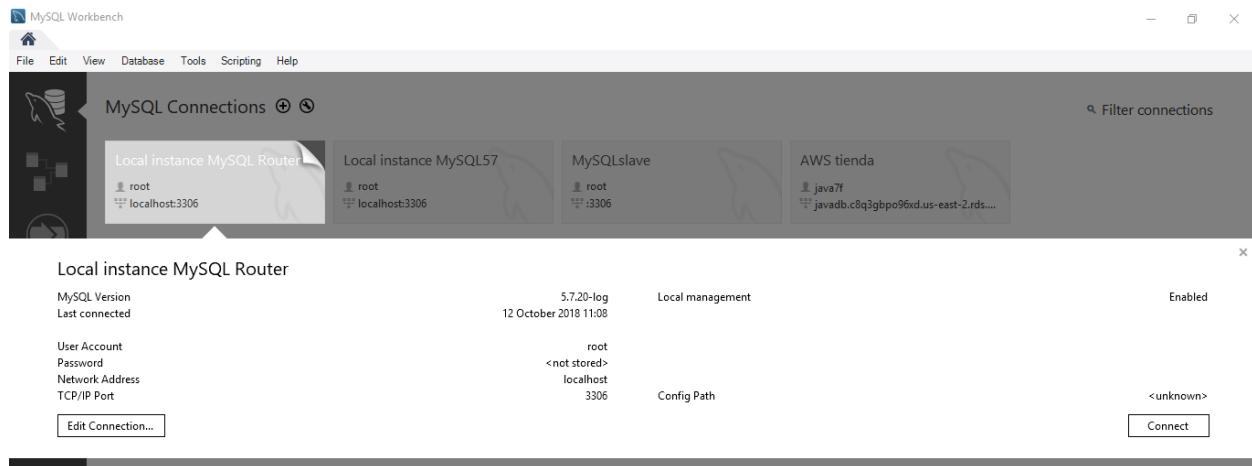
Failover de una base de datos

Durante la clase hemos aprendido los distintos tipos de requerimientos. Partiendo de esto, definimos a los atributos de calidad a todos aquellos requerimientos que no sean funcionales. Para la siguiente práctica se nos pidió la elaboración de un programa hiciera uso de bases de datos para su funcionamiento, con el objetivo de darle al sistema el atributo de calidad de alta disponibilidad, a través del uso de *failover* de una base de datos.

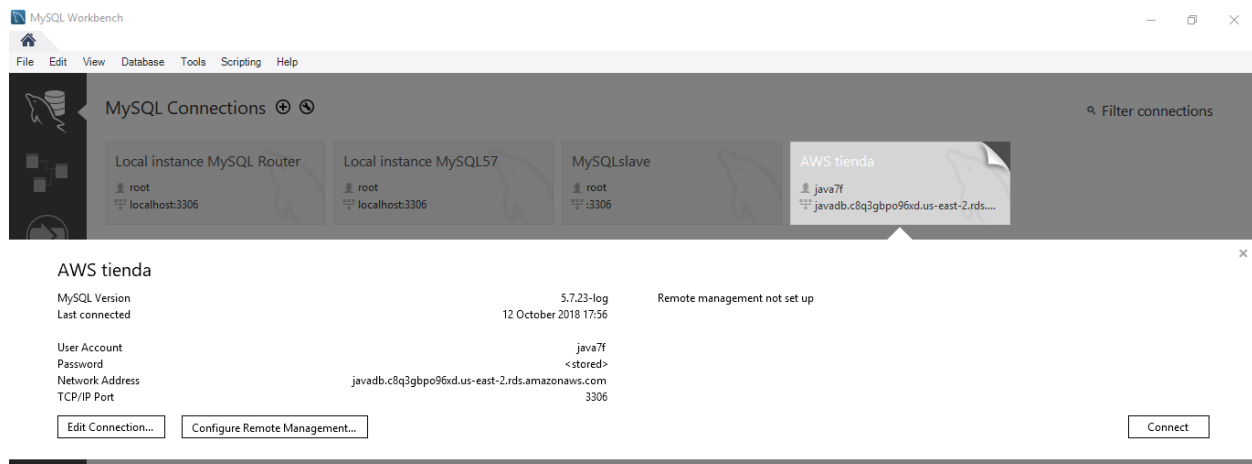
Para la realización de esta práctica reutilicé el proyecto de Tienda Virtual que fue desarrollado en materias anteriores. Dicho proyecto consiste en una tienda digital de ventas de celulares y computadoras, la cual utiliza una base de datos para el registro de los productos en inventario. Con unas adaptaciones al código, logré hacer que manejara las conexiones a una base de datos local y una remota, alojada en servidores de AWS.

El funcionamiento básicamente es el siguiente: tenemos dos instancias de bases de datos, una local ("Tienda") y otra remota ("Tienda2"), la cual está alojada en los servidores de Amazon Web Service. Esta modalidad permite que el programa tenga dos puntos de acceso independientes, evitando tener una dependencia a un solo servidor. El programa, por defecto, está conectado a ambas bases de datos de manera simultánea, de manera de que todas las consultas realizadas en tiempo de ejecución sean aplicadas en ambas bases de datos. Esto es con el objetivo de mantener la sincronización de los datos. Ahora bien, ¿qué pasa cuando una de las bases está caída? Cuando este evento ocurre, la base de datos que permanece activa toma la completa responsabilidad del funcionamiento del programa, haciéndose cargo de todas las consultas que se hagan. Como el sistema chequea las conexiones cada vez que se ejecuta un *query*, siempre se sabe en qué estado se encuentran las bases. Dicho esto, ¿qué pasaría cuando la base de datos caída estuviera disponible de nuevo? Mediante el uso de archivos, el programa tendría la capacidad de sincronizar los datos para que ambas bases de datos tuvieran la misma información. Los archivos contienen las consultas que fueron ejecutadas mientras la base de datos estuvo abajo y, a través de variables booleanas, el programa sabría cuándo tocaría actualizar las bases para luego blanquear los archivos.

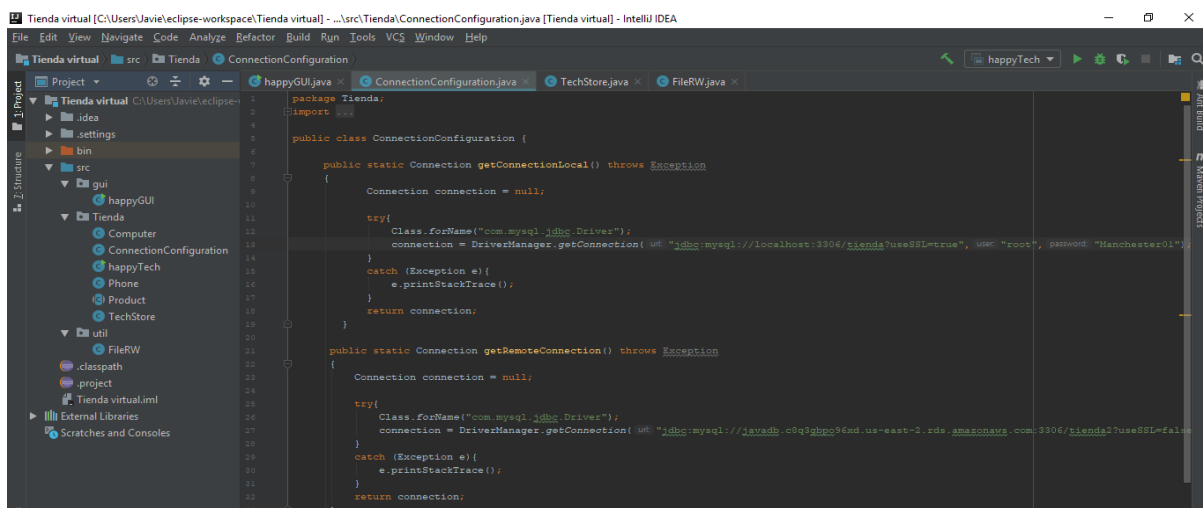
- Base de datos local



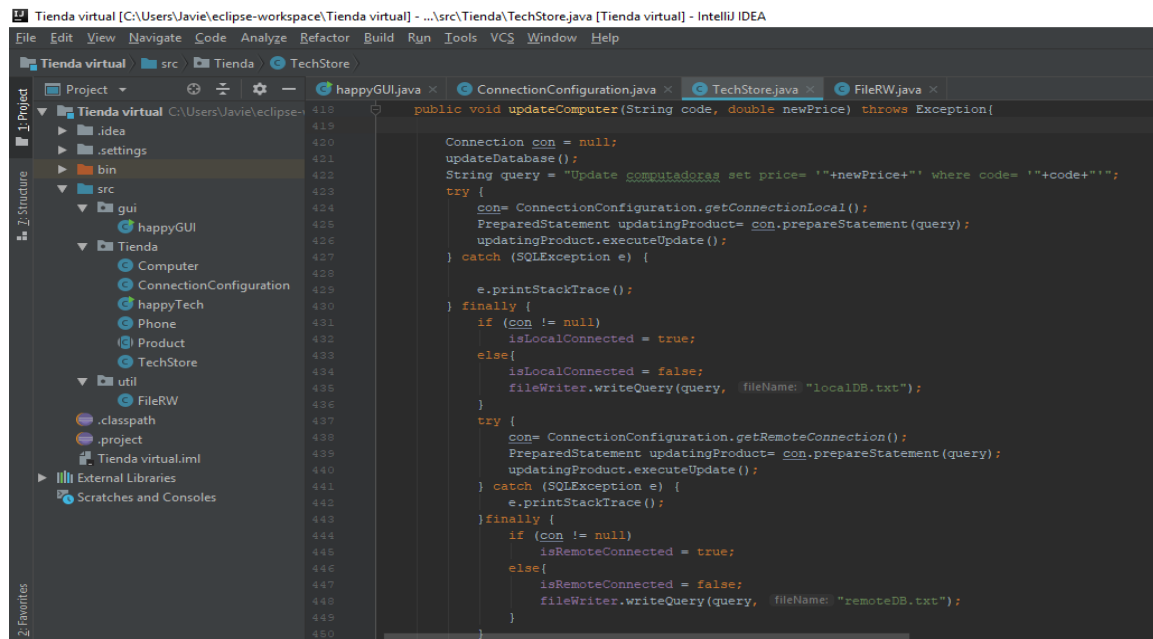
- Base de datos remota



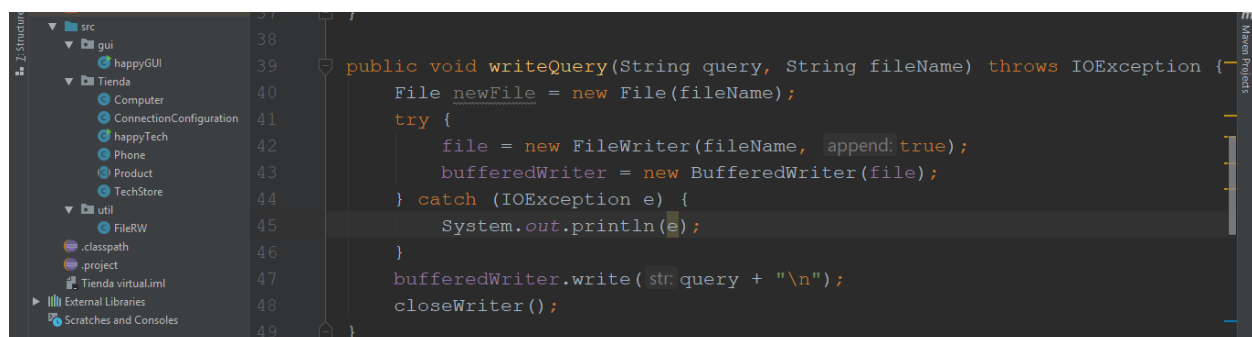
- Función que obtiene las conexiones para ambas bases de datos.



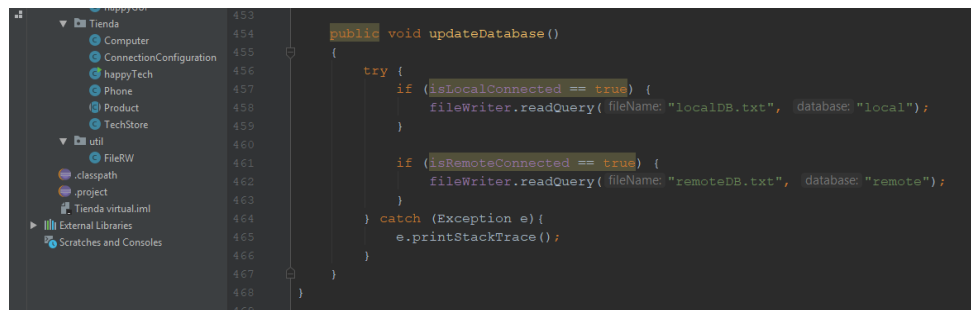
- Ejemplo de la ejecución de consultas. Cuando las dos bases están arriba, las consultas son ejecutadas en ambas. Podemos ver cómo el programa se maneja cuando alguna conexión fue nula: se coloca la variable booleana correspondiente a la base afectada en falso, para indicar que está caída y, luego, se llama a la función de *writeQuery()* para que registre en el archivo de texto la consulta a ejecutar.



- Vista de la función de escritura contenida en la clase *FileRW*



- Vista de la función que actualiza los datos de cada base



```
453 public void updateDatabase()
454 {
455     try {
456         if (isLocalConnected == true) {
457             fileWriter.readQuery( fileName: "localDB.txt", database: "local");
458         }
459         if (isRemoteConnected == true) {
460             fileWriter.readQuery( fileName: "remoteDB.txt", database: "remote");
461         }
462     } catch (Exception e){
463         e.printStackTrace();
464     }
465 }
```