Para la realización de estas actividades podría ser necesario utilizar clases, métodos u opciones no vistos en el capítulo, por lo que se recomienda consultar la documentación de Java SE 8 (https://docs.oracle.com/javase/8/docs/api/)

Hay que utilizar transacciones para las modificación de los datos siempre que sea apropiado.

NOTA: se entregará un programa Java para cada uno de los apartados.

1. Haz un programa que permita navegar de forma interactiva por los contenidos de la tabla **CLIENTES** creada en los ejemplos de las transparencias del tema (y que encontrarás al pie de este ejercicio). Primero, el programa debe realizar una consulta para obtener los contenidos de la tabla, y debe mostrar el mensaje "fila 1" y el contenido de la fila, indicando para cada columna el nombre de la columna y su valor. Después, se deben ejecutar los comandos que se vayan introduciendo por teclado. Si el comando es ".", debe terminar, por supuesto liberando todos los recursos. Si es "k", debe ir a la siguiente fila, indicar el número de la fila y mostrar sus contenidos, como al principio para la primera fila. El comando para ir a la fila anterior será "d". Si se introduce un número, se debe mostrar la fila en la posición indicada por el número. El programa debe mostrar mensajes apropiados en caso de que el comando que se ha introducido no se puede realizar (por ejemplo, estando en la última fila se pide ir a la siguiente, o se introduce el número de una fila que no existe). La clase **Integer** tiene métodos que permiten determinar si un **String** representa un número entero.

   Nota: se puede leer una cadena de caracteres desde teclado de la siguiente forma:

   ```
   BufferedReader br =new BufferedReader(new InputStreamReader(System.in));
   String comando = br.readLine();
   ```

   ```
   #Tabla clientes
   CREATE TABLE `clientes` (`DNI` char(9) NOT NULL,`APELLIDOS`
   varchar(32) NOT NULL, `CP` char(5) DEFAULT NULL,
     PRIMARY KEY (`DNI`));
   ```

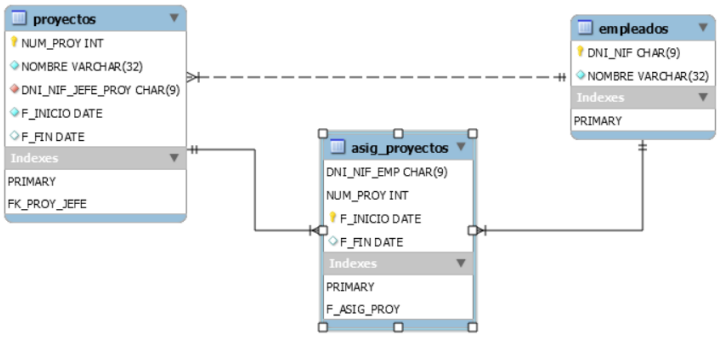
2. Mejora el programa anterior para que se pueda utilizar con cualquier tabla, cuyo nombre se pedirá al iniciarse el programa. Será necesario obtener información acerca de la tabla mediante el método **getMetaData()** de **ResultSet.**

   Para algunos ejercicios que siguen, debe utilizarse un conjunto de tablas para representar los proyectos que desarrollan una empresa, sus

empleados y las asignaciones de empleados a proyectos. Las tablas se deben crear en MySQL con las siguientes sentencias (una posibilidad es ejecutarlas con un programa Java con opción de conexión **allowMultiQueries=true**).

```
CREATE TABLE EMPLEADOS(DNI_NIF CHAR(9) NOT NULL, NOMBRE VARCHAR(32) NOT
      NULL, PRIMARY KEY(DNI_NIF));
CREATE TABLE PROYECTOS(NUM_PROY INTEGER AUTO_INCREMENT NOT NULL, NOMBRE
      VARCHAR(32) NOT NULL, DNI_NIF_JEFE_PROY CHAR(9) NOT NULL, F_INICIO
      DATE NOT NULL, F_FIN DATE, PRIMARY KEY(NUM_PROY), FOREIGN KEY
      FK_PROY_JEFE(DNI_NIF_JEFE_PROY) REFERENCES EMPLEADOS(DNI_NIF));
CREATE TABLE ASIG_PROYECTOS(DNI_NIF_EMP CHAR(9), NUM_PROY INTEGER NOT
      NULL, F_INICIO DATE NOT NULL, F_FIN DATE, PRIMARY
      KEY(DNI_NIF_EMP,NUM_PROY, F_INICIO), FOREIGN KEY
      F_ASIG_EMP(DNI_NIF_EMP) REFERENCES EMPLEADOS(DNI_NIF), FOREIGN
      KEY F_ASIG_PROY(NUM_PROY) REFERENCES PROYECTOS(NUM_PROY));
```

Para mayor claridad, el siguiente diagrama, obtenido con MySQL Workbench, muestra los esquemas de las tablas, incluyendo campos y claves primarias, y las relaciones entre tablas (claves foráneas).



3. Crea una clase **GestorProyectos** que contenga métodos para almacenar datos de empleados, proyectos y asignaciones de empleados a proyectos. La clase debe tener métodos para:
    a. Crear un nuevo empleado (**nuevoEmpleado**). Debe devolver true si el empleado se creó correctamente. A este método se le pueden pasar todos los datos del empleado. Un valor *null* para alguno significa que no se especifica valor. No debe validar los datos que se le pasan. Si el valor indicado para alguno provoca que se lance alguna excepción, este método la propagará. Es decir, su definición debe incluir la opción **throws** con la clase de excepción correspondiente (al menos **SQLException**).
    b. Crear un nuevo proyecto (**nuevoProyecto**). Debe devolver el número de proyecto (**NUM_PROY**). Como el método anterior, tiene parámetros para todos los datos del proyecto (un valor *null* significa que no se especifica valor), no valida los valores proporcionados para ellos, y propaga excepciones. Si se especifica *null* para

F_INICIO, debe asignarse la fecha actual como fecha de inicio del proyecto, que en MySQL se puede obtener con la función **now()**. Un valor *null* para F_FIN significa que no está informada, y debe asignarse un valor NULL en la base de datos.

c. Asignar un empleado a un proyecto (**asignaEmpAProyecto**). Seguir para ello las mismas directrices que para los métodos anteriores, incluyendo las referentes a F_INICIO y F_FIN.

Debe probarse esta clase mediante un programa de prueba en el método **main()** que cree varios empleados y proyecto, y realice la asignación de algunos empleados a proyectos. No es necesario realizar ninguna verificación de fechas. Por ejemplo: que la fecha de inicio de una asignación de un empleado a un proyecto no es anterior a la fecha de inicio del proyecto, y otras similares. Pero por supuesto se pueden incluir como mejora. Una posibilidad es realizar estas verificaciones en la clase de Java. Otra es realizarlas mediante restricciones de integridad definidas en la propia base de datos o mediante *triggers*.

4. Se trata de hacer algo similar a lo hecho en el anterior ejercicio pero con un planteamiento algo distinto. Hay que crear clases **Empleado**, **Proyecto** y **AsignaciónEmpAProyecto**. Cada una de ellas debe tener un constructor sin parámetros y campos correspondientes a los campos de las correspondientes tablas en la base de datos. Deben tener métodos **getXXX()** y **setXXX()** para cada propiedad. Por ejemplo, para "Empleado" serían **String getDNINIF(), void getDNINIF(String DNINIF), String getNombre() y void setNombre(String nombre)**. Aparte del constructor sin parámetros, debe tener uno con los parámetros correspondientes a los campos de la clave primaria. Por ejemplo: Empleado (String DNINIF), que lanzará un excepción **SQLException** si no existe en la base de datos una fila para los valores de atributos de la clave primaria proporcionados. Deben tener un método **save()** que guarde el objeto en la base de datos, con una sentencia **INSERT**, si no existe (en el caso de clientes, si no existe ninguno con el DNI), o que modifique los datos, con una sentencia **UPDATE**, si existe. Puedes considerar el uso de **INSERT … ON DUPLICATE KEY UPDATE** en MySQL, o sentencias similares en otras bases de datos.

5. Completa la clase Proyecto desarrollada en el ejercicio 4 con un método **getListAsigEmpleados()** que devuelva una lista con los empleados asignados actualmente al proyecto. Un empleado está asignado actualmente a un proyecto si existe para él una fila en **ASIG_PROYECTOS** con **F_INICIO** anterior a la fecha actual (se puede recuperar en MySQL con **now()** y con F_FIN no informada (es decir, con valor null) o posterior a la fecha actual.

6. Haz un programa para insertar en una tabla de clientes los datos leídos de un fichero en formato CSV, que debes preparar tú mismo. Los datos de cada cliente deben estar en una línea distinta, y como separador de campos puedes utilizar ";" o "|". Debe crearse una clase **LectorDatosClientes,** con un método **insertarDatosClientes(String nombreFichero, String nombreTabla, String separadorCampos).** La tabla debe tener la misma estructura que la tabla de ejemplo CLIENTES, y debe estar creada previamente a la ejecución del programa. El parámetro **separadorCampos** indica el separado de campos. Se puede leer el fichero línea a línea utilizando un **BufferedReader**. Se puede obtener los campos de una línea utilizando la clase **StringTokenizer**, o bien con el método **split()** de String. Se debe utilizar una sentencia preparada (**PreparedStatement**) para las operaciones INSERT, y deben agruparse todas en una única transacción. Si el carácter separador aparece dos veces seguidas, eso significa que para un campo no se ha especificado un valor, y entonces se deben asignar el valor *null*. Haz alguna prueba con ficheros cuyos contenidos puedan provocar errores, para asegurarte de que se da el mensaje de error apropiado, y no se realiza ningún cambio, al estar todas las operaciones dentro de una transacción. Por ejemplo, especifica un valor nulo para DNI o valores incorrectos para algunos campos.

7. Haz una lista lo más completa posible con condiciones de prueba para el programa del ejercicio 6. Para cada condición de prueba, indica el resultado esperado. Un ejemplo de condición de prueba sería:"Se especifica valor nulo para tal columna (que no admite valor nulo)", y el resultado esperado: "se muestra un mensaje de error apropiado, se termina la ejecución del programa y no se realiza ningún cambio en la base de datos". Otra sería: "Se especifica un valor alfabético para tal columna (que tiene valores numéricos)", y resultado esperado podría ser el mismo que para la anterior condición de prueba. Verifica cada condición de prueba con un fichero apropiado. No es necesario que el programa proporcione mensaje de error específico, ni que en las condiciones de prueba se indique un error específico. Normalmente bastará con que el programa gestione las excepciones de tipo **SQLException** y muestre la información que proporciona esta clase de excepciones. Se recomienda redactar las condiciones de prueba sin tener el programa en mente, como lo haría alguien que no sabe cómo está hecho el programa, pero sí qué debe hacer.