Estructura de Tablas

Dada la tabla de recorridos:

> ### *Recorridos (idRecorrido(PK), estacionOrigen, estacionDestino, kms, horaSalida, horaLlegada, precio)*
>
> Que representa un recorrido del tren que podría repetrise en varias fechas, caracterizado por una estación origen otra destino, una hora de salida y otra de llegada.
>
> Como en Oracle no hay campos TIME las horas de salida y llegada se han representado mediante campos TIMESTAMP en los que despreciaremos la fecha.

la tabla de viajes:

> ### *Viajes (idViaje(PK), idTren, idRecorrido --> FK a recorridos, fecha, nPlazasLibres, realizado, idConductor)*
>
> que representa el viajes que hace un tren en un determinado recorrido y en una determinada fecha.
>
> ***`NplazasLibres`*** representa el número de plazas libres que quedan en el tren.
>
> ***`realizado`*** es un booleano que indica si el viaje se ha realizado o no, y el ***`idConductor`*** identifica al conductor (no intervienen en este problema).

la tabla de tickets

> ### *Tickets (idTicket(PK), idViaje (FK ---> Viajes, fechaCompra, cantida, precio)*
>
> Que representa un ticket o un conjunto de varios tickets (por eso hay un campo ***`cantidad`*** que indica cuantos) correspondientes a un viaje.
>
> El campo ***`precio`*** contiene la multiplicación del campo ***`cantidad`*** por el campo ***`recorrido.precio`***, correspondiente al recorrido de ese viaje.

El script que crea las tablas está en `sql/CompraBilleteTren.sql` y el *main* ya lo corre automáticamente al llamar a *`pruebaCompraBilletes()`*, que a su vez llama a *`creaTablas()`*.

El resto de tablas del script no intervienen en este problema.

Descripción General

En `lsi.ubu.enunciado.EsqueletoCompraBilleteTren.java` está el fichero que se te pide completar. Del mismo, se pide implementar el método:

```
public static void comprarBillete( Time p_hora, java.util.Date p_fecha, String
p_origen, String p_destino, int p_nroPlazas) throws SQLException
```

que dada una hora, una fecha, una ciudad origen y una ciudad destino que sirven para caracterizar un viaje, inserta una fila en la tabla de *tickets* por la compra de `p_nroPlazas`, decrementando el numero de plazas libres de ese viaje.

Asimismo, habrá que completar `lsi.ubu.enunciado.CompraBilleteTrenException`

Algoritmo

Para ello, se sugiere seguir estos pasos (pero se puede hacer de otras formas):

1. Consultar el ***id_viaje*** correspondiente a la fecha `p_fecha`, origen `p_origen`, destino `p_destino` y hora de salida `p_hora` pasados por parámetro. Es recomendable que en la misma consulta obtengasel ***precio*** que tiene el recorrido de ese viaje.

2. Decrementar en la tabla de viajes el número de plazas libres ***NplazasLibres*** según el parámetro `p_nroPlazas`.

3. Insertar el ticket correspondiente, teniendo en cuenta que el ***idTicket*** sale de la secuencia `seq_tickets` y que la fecha de compra es la del sistema.

El método tomará una conexión del pool al principio, y cometerá la transacción. Al finalizar cerrará todos los recursos utilizados.

Excepciones

El método lanza sólo excepciones del tipo `CompraBilleteTrenException` que son dos posibles (Ver `lsi.ubu.enunciado.CompraBilleteTrenException`.):

1. La primera registrará el problema de que **no queden plazas suficientes para ese viaje**. Su código será el "1" y el mensaje de error `"No hay plazas suficientes."`.

2. La segunda registrará el problema de que **no exista un viaje que corresponda con ese origen, destino, fecha y hora de salida**. Su código será el "2" y el mensaje de error `"No existe ese viaje para esa fecha, hora, origen y destino."`.

El resto de excepciones serán tratadas dentro del propio método.

Toda excepción sea del tipo que sea retrocederá la transacción y se lanzarán o propagarán al main que hace la llamada al método.

Las excepciones **que no sean** del tipo `CompraBilleteTrenException` dejará el mensaje del error con `logger.error(...)` en el *logger*.

Pruebas automáticas

Las pruebas automáticas se invocan desde el *main* y están en el método `pruebaCompraBilletes` . Aunque no hayas empezado a hacer nada el fichero `EsqueletoCompraBilleteTren.java` compila y ya te muestra los mensajes de las pruebas aún no superadas. Esas pruebas hacen 3 cosas

1. Comprar un billete para un viaje inexistente, (de momento, al ejecutarlo, te sale `NO se da cuenta de que no existe el viaje MAL`).

2. Comprar un billete de 50 plazas en un vieje que no tiene tantas plazas libres. (por eso, de momento, al ejecutarlo, te sale `NO se da cuenta de que no hay plazas MAL`).

3. Finalmente hacemos una compra de un billete de 5 plazas sin problemas. De momento te saldrá `Compra ticket MAL` a la espera que programes correctamente el método.

El mesaje que sále cuando todo funciona (salvo por la fecha y hora de la excepción que se registra en el *logger*) es:

```
2018-03-21 19:16:08 ERROR CompraBilleteTrenException:41 - No existe ese viaje para esa
fecha, hora, origen y destino.
Se da cuenta de que no existe el viaje OK
2018-03-21 19:16:08 ERROR CompraBilleteTrenException:41 - No hay plazas suficientes.
Se da cuenta de que no hay plazas OK
Compra ticket OK
FIN............
```

Valoracion de la practica

La realizacion completa de este enunciado supone 7.5 puntos sobre 10 de la nota total, para la optencion de los otros 2.5 puntos sobre 10 hay que realizar el metodo

```
public static void anularBillete( Time p_hora, java.util.Date p_fecha, String
p_origen, String p_destino, int p_nroPlazas) throws SQLException
```

El método lanza sólo excepciones del tipo `CompraBilleteTrenException` que hay que añadir a las que ya existen. Hay que implementar pruebas automaticas que demuestren su correcto funcionamiento similares a las existentes para `comprarBillete.`

La practica se debe presentar en un solo ZIP y contener una sola carpeta que se pueda descomprimir y que usando la función de Eclipse “Open Projects from File System” importe y se pueda ejecutar. La solucion debe estar en `lsi.ubu.solucion` tanto el fichero `CompraBilleteTren.java` como el `CompraBilleteTrenException` Comprobar que esto es así ya que no se corregirá ni puntuara una práctica que no se importe y ejecute correctamente.

Tambien hay que añadir en la carpeta enunciado un PDF con los participantes en la practica. Si ya alguno quiere rematarlo puede añadir un PDF con una breve explicacion del codigo añadido para la realizacion del metodo `anularBillete.`