ESCUELA PROFESIONAL DE INGENIERIA DE SISTEMAS

**GUIA DE LABORATORIO NRO. 07**

|  |
| --- |
| **ESCUELA :** **INGENIERIA DE SISTEMAS**  **ASIGNATURA :** **GESTIÓN DE DATOS E INFORMACIÓN II**  **CICLO**  : **V**  **TURNO** : MAÑANA/TARDE/NOCHE  **SEMESTRE** : 2022-1  **DOCENTE**  : Dr.Ivan Petrlik Azabache |

1. **TEMA:** Transacciones
2. **OBJETIVOS:**

* Desarrollar los ejercicios propuestos por el docente.

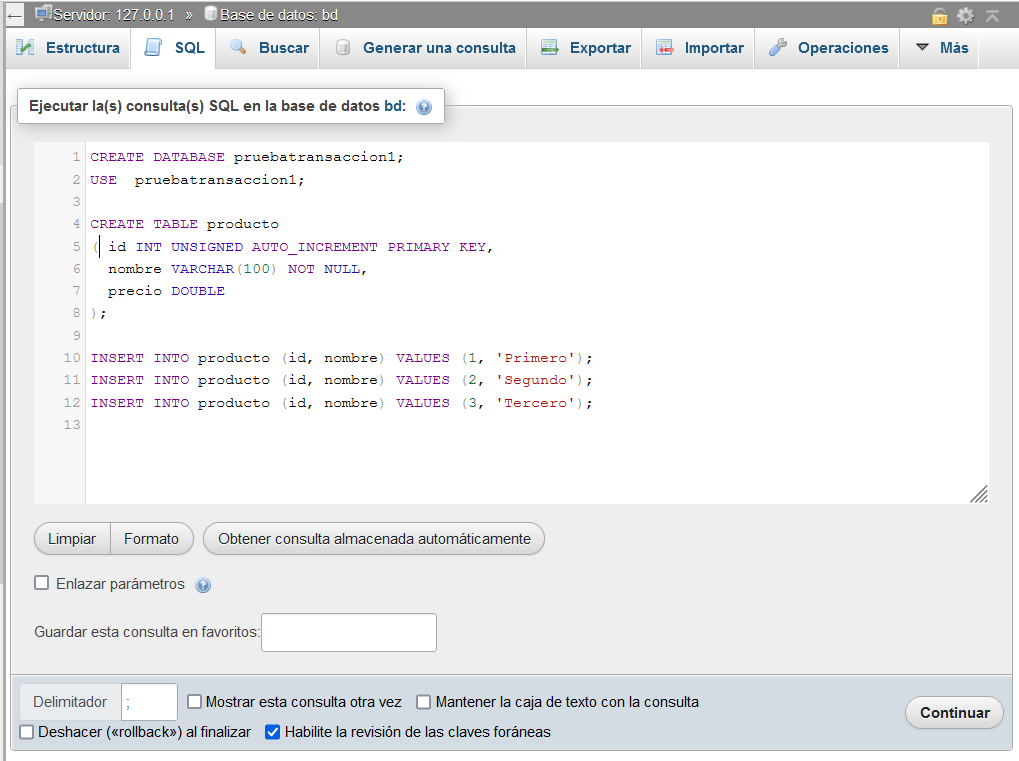
1. **METODOLOGÍA:**

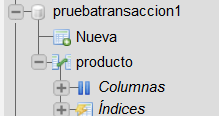
* Solución de los ejercicios propuestos con la guía del docente.
* Realizar un informe de toda la secuencia del laboratorio propuesto utilizando pantallas capturadas de los ejercicios propuestos explicando y justificando los resultados.

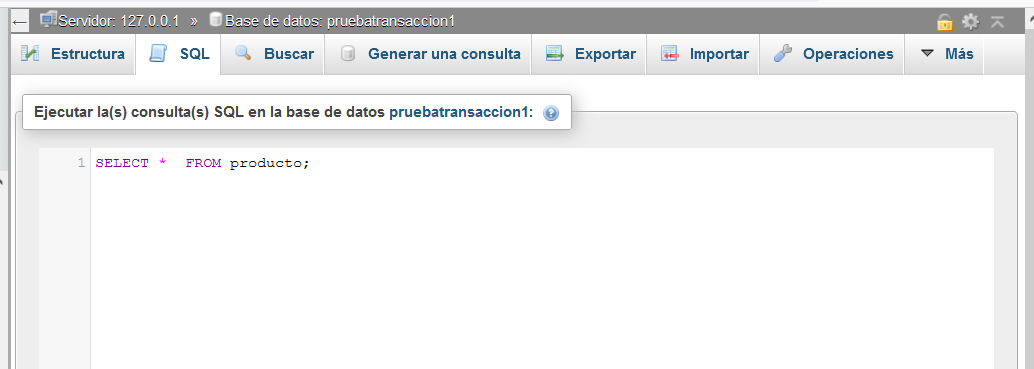
1. **RECURSOS:**
2. Equipo computacional con conexión a internet.
3. Utilizar el software SQL Server
4. Utilizar la plataforma Blackboard para entregar el laboratorio
5. Utilizar la plataforma de Zoom para la sustentación de laboratorio.
6. **DESARROLLO DE ACTIVIDADES:**
   1. Descripción
   2. Proceso

**Ejercicios prácticos de transacciones**

1. Ejecuta las siguientes instrucciones y resuelve las cuestiones que se plantean en cada paso.

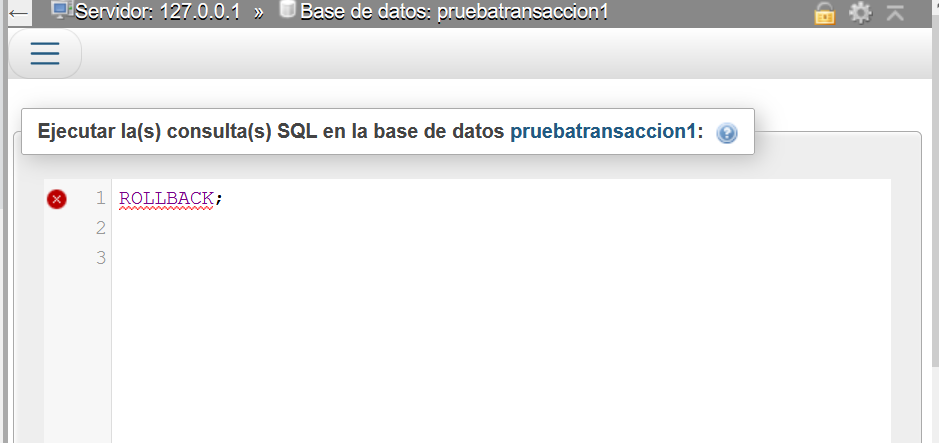




**1. ¿Qué devolverá esta consulta?**

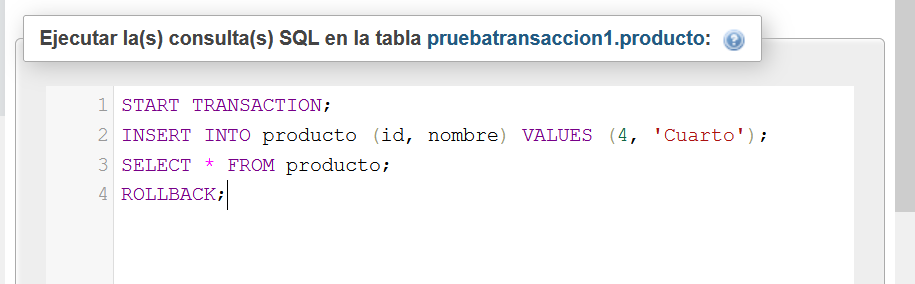
**Responder de manera textual y adjuntar la evidencia de captura de pantalla de lo realizado.**

1. **Vamos a intentar deshacer la transacción actual y luego de ejecutar el ROLLBACK , realizar una consulta a la tabla producto . Explique lo observado**



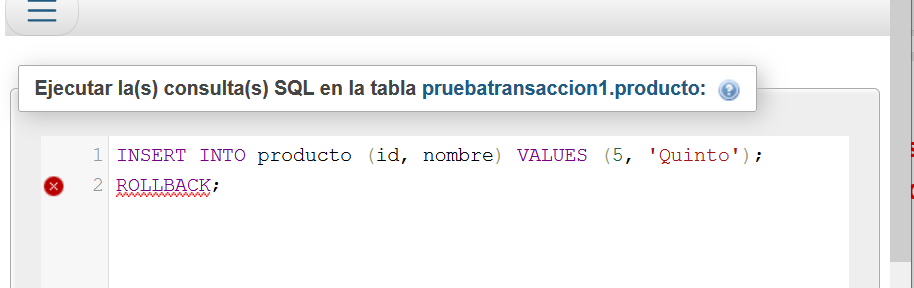
**Responder de manera textual (Justifique su respuesta ) y adjuntar la evidencia de captura de pantalla de lo realizado.**

**3) Ejecutamos la siguiente transacción y luego de esta realice nuevamente un select \* from producto.**



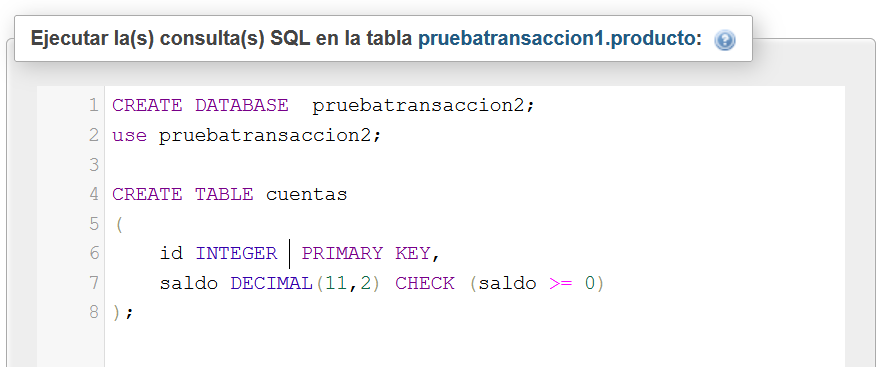
**Responder de manera textual ( Justifique su respuesta ) y adjuntar la evidencia de captura de pantalla de lo realizado.**

**4) Ejecutamos la siguiente transacción y luego de esta mismas realice una consulta a la tabla producto.**

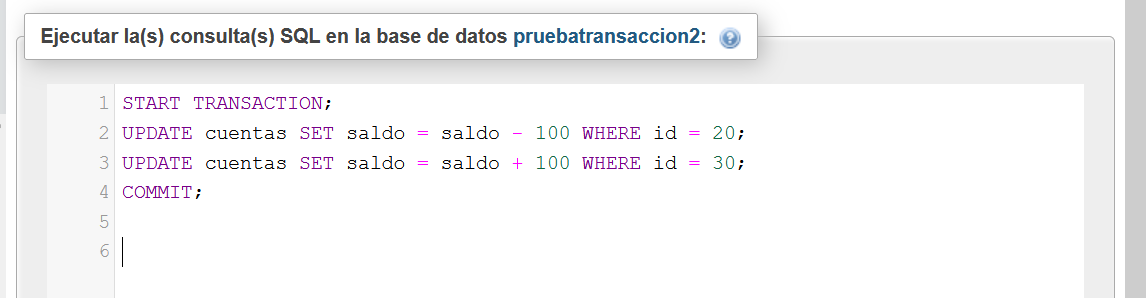


**Responder de manera textual (Justifique su respuesta) y adjuntar la evidencia de captura de pantalla de lo realizado.**

**5) Considera que tenemos una tabla donde almacenamos información sobre cuentas bancarias definida de la siguiente manera:**



**Suponga que queremos realizar una transferencia de dinero entre dos cuentas bancarias con la siguiente transacción:**



**Responder de manera textual (Justifique su respuesta) y adjuntar la evidencia de captura de pantalla de lo realizado.**

* **¿Qué ocurriría si el sistema falla o si se pierde la conexión entre el cliente y el servidor después de realizar la primera sentencia UPDATE?**
* **¿Qué ocurriría si no existiese alguna de las dos cuentas (id = 20 y id = 30)?**
* **¿Qué ocurriría en el caso de que la primera sentencia UPDATE falle porque hay menos de 100 € en la cuenta y no se cumpla la restricción del CHECK establecida en la tabla?**

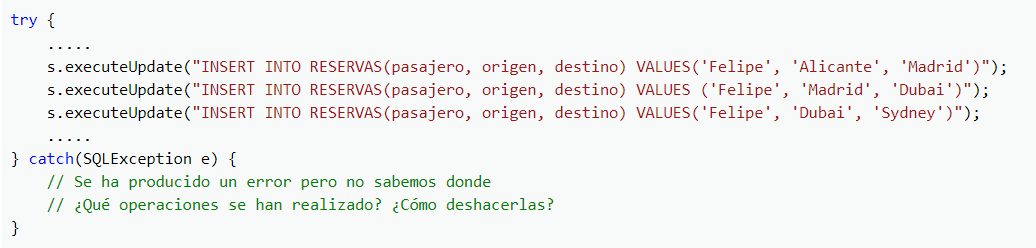
**Leer teórica de Transacciones en Java**

Una transacción es un conjunto de operaciones sobre una base de datos que se deben ejecutar como una unidad.

Hay ocasiones en las que es necesario que varias operaciones sobre la base de datos se realicen en bloque, es decir, que se ejecuten o todas o ninguna, pero no que se realicen unas sí y otras no.

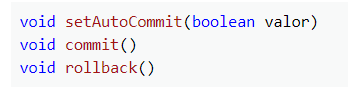
Si se ejecuta parcialmente hasta que da error, el estado de la base de datos puede quedar inconsistente. En este caso necesitaríamos un mecanismo para devolverla a su estado anterior, pudiendo deshacer todas las operaciones realizadas.

Como ejemplo, supongamos que vamos a reservar un vuelo desde Alicante hasta Sydney. Para llegar a Sydney tenemos que realizar varias escalas. Tendremos que reservar un vuelo de Alicante a Madrid, otro de Madrid a Dubai y otro desde Dubai a Sydney. Si representamos cada reserva como un insert en la base de datos tendremos las siguientes instrucciones:



Un objeto Connection por defecto realiza automáticamente cada operación sobre la base de datos. Esto significa que cada vez que se ejecuta una instrucción, se refleja en la base de datos y no puede ser deshecha. Por defecto está habilitado el modo auto-commit en la conexión.

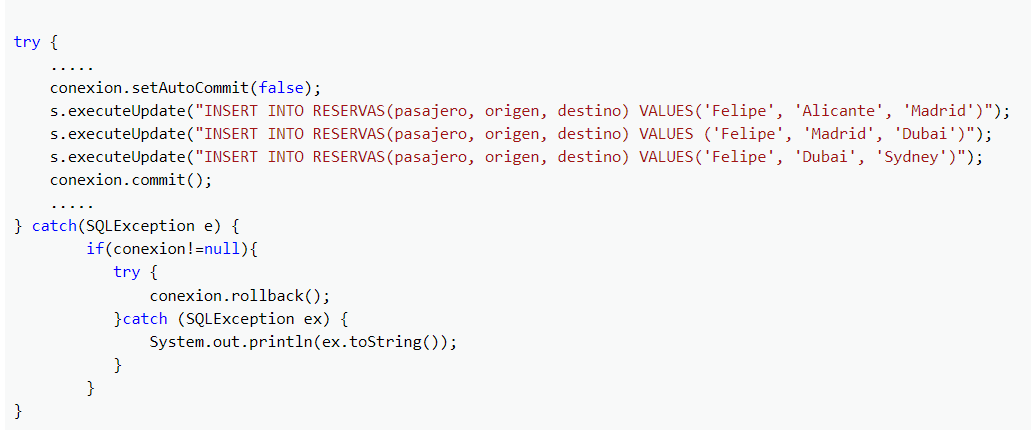
Los siguientes métodos en la interfaz Connection son utilizados para gestionar las transacciones en la base de datos:



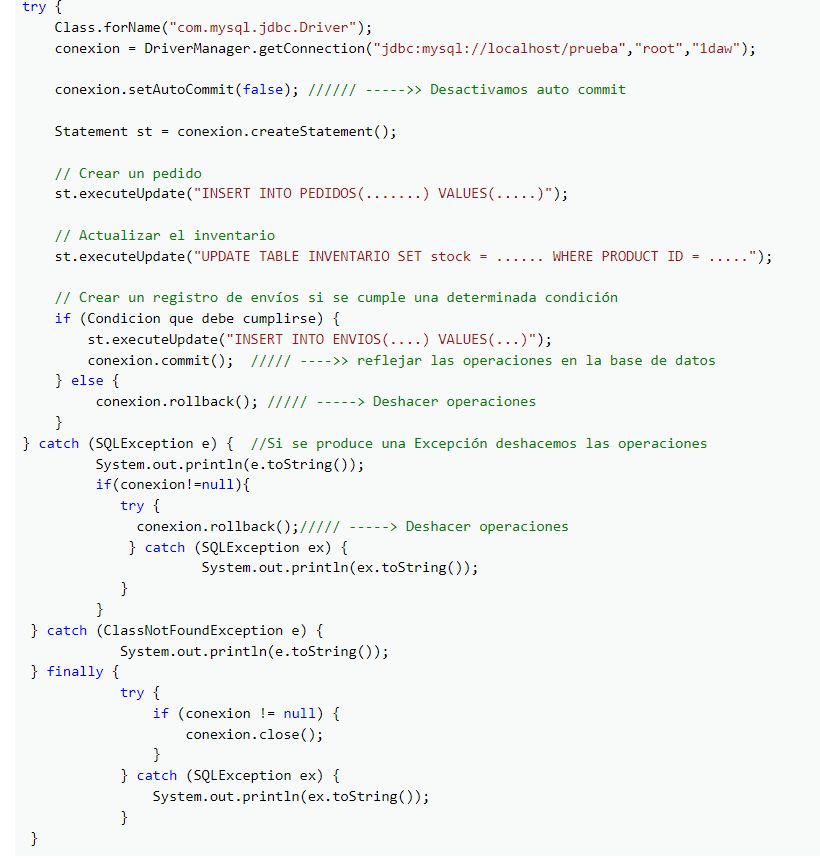
Para iniciar una transacción deshabilitamos el modo auto-commit mediante el método setAutoCommit(false) . Esto nos da el control sobre lo que se realiza y cuándo se realiza.

Una llamada al método commit() realizó todas las instrucciones emitidas desde la última vez que se invocó el método commit() .

Una llamada a rollback() deshará todos los cambios realizados desde el último commit() . Una vez se ha publicado una instrucción commit() , esas transacciones no pueden destruirse con rollback() .



Otro ejemplo de transacciones: Tenemos un fragmento de código que crea un pedido, actualiza el inventario y crea un registro de envíos. Si falla la creación de un registro de envíos no se debe crear el pedido. En ese caso, los efectos de las SQL correspondientes a las dos primeras tareas (crear un pedido y actualizar el inventario) deben desaparecer.



Transacciones – Uso de Commit y Rollback en Java

Cuando desarrollamos aplicaciones en java con base de datos debemos tener cuidado en que se cumplan un conjunto de características conocida como ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), en el caso de este post nos centraremos en la Atomicidad y Consistencia

Atomicidad: es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias. Se dice que una operación es atómica cuando es imposible para otra parte de un sistema encontrar pasos intermedios. Si esta operación consiste en una serie de pasos, todos ellos ocurren o ninguno. Por ejemplo, en el caso de una transacción bancaria o se ejecuta tanto el depósito como la deducción o ninguna acción es realizada.

Consistencia: Integridad. Es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto, se ejecutan aquellas operaciones que no van a romper las reglas y directrices de integridad de la base de datos. La propiedad de consistencia sostiene que cualquier transacción llevará a la base de datos desde un estado válido a otro también válido. «La Integridad de la Base de Datos nos permite asegurar que los datos son exactos y consistentes, es decir que estén siempre intactos, sean siempre los esperados y que de ninguna manera cambien ni se deformen. De esta manera podemos garantizar que la información que se presenta al usuario será siempre la misma

**Planteando una situación para este tutorial**

Si tenemos una serie de sentencias **SQL INSERT** por ejecutar y por un motivo *X ,* surge un error que no permite que se completen todas estas sentencias nos encontramos con el problema de que hasta el punto que se produjo el error, los *INSERT* se ejecutaron y se guardaron nuevos registros en la base de datos, entonces estaríamos violando los criterios de Atomicidad y de Consistencia porque no primeramente no se completo la transacción en su totalidad pero tenemos nuevos registros en la base de datos pero estos registros están incompletos.

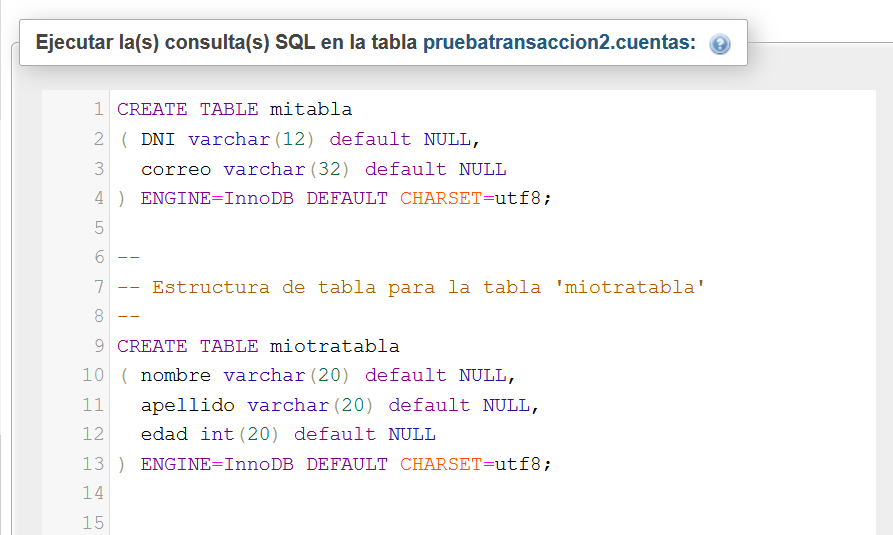
**¿Como podemos solucionar este problema?**

**Necesitamos**

* Java
* IDE Netbeans 12.4
* Libreria MySQL JDBC Driver
* MySQL

**Base de datos.**

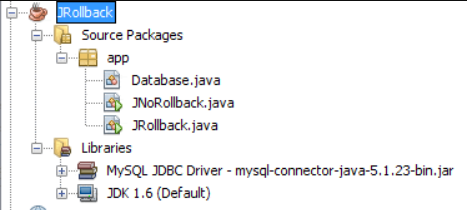
Para el ejemplo de este post, utilizaremos dos tablas «**miTabla**» y «**miOtraTabla**«, su estructura es la siguiente:



**Nota importante :** Al crear las tablas, debemos utilizar **ENGINE=InnoDB** y no **MyISAM** esto porque *InnoDB* tiene soporte para transacciones, bloqueo de registros y nos permite tener las características ACID  garantizando la integridad de nuestras tablas.

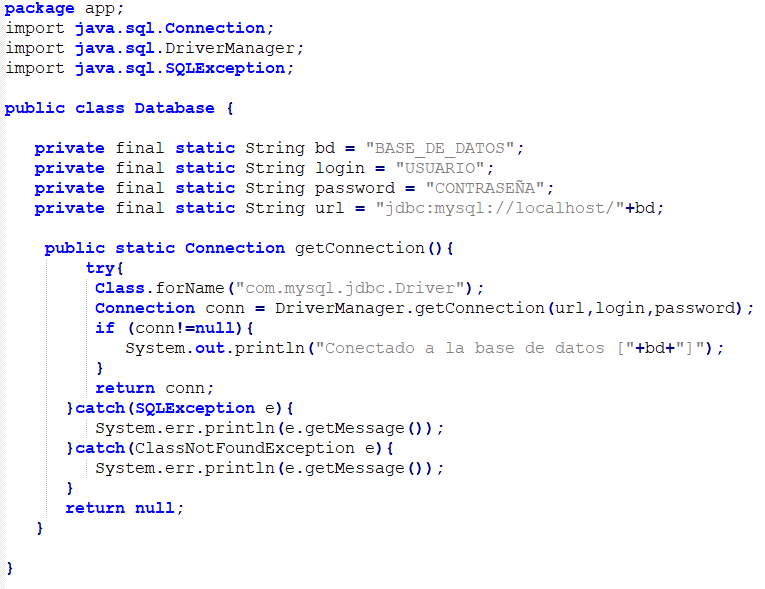
**Proyecto Netbeans**

Nuestro proyecto base será el siguiente:



* Le añadimos el conector **JDBC** al proyecto
* Tenemos una clase llamada**Database.java** que nos permite conectarse a la base de datos MySQL
* Tenemos dos clases Main, **JNoRollback y JRollback**, en la primera clase se implementara las instrucciones INSERT sin el uso de *Commit* y *Rollback* , en la segunda clase con el uso de estos 2 métodos, el fin es apreciar mejor que pasa cuando se produce un error al ejecutar estas transacciones.

**Clase Database.java**



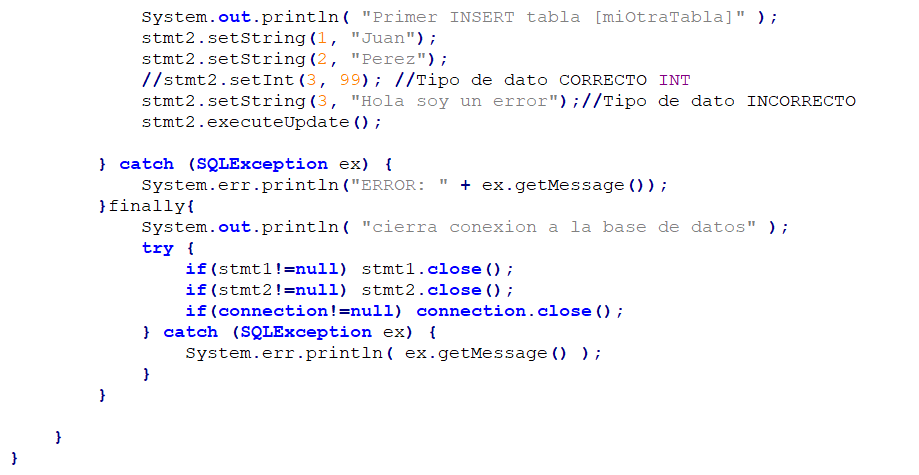
**Nota importante:**

Terminado el laboratorio, el docente designara 10 estudiantes para que compartan pantalla y verificar el cumplimiento de la actividad del día de hoy.

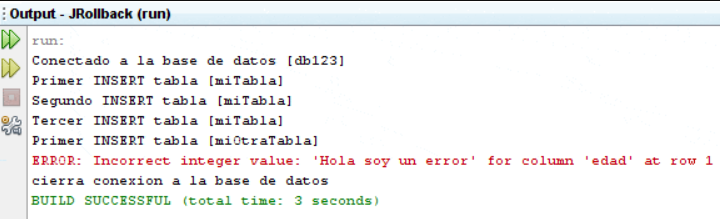
**Clase JNoRollback.java**

En esta clase no se hace uso de los métodos *Commit* y *RollBack* por lo que las instrucciones **INSERT** se irán ejecutando una tras otra afectando directamente a la base de datos, si no se encontrara ningún error todo terminaría normalmente, sin embargo en la linea de código 46 intencionalmente se intercambia el tipo de dato INT por un STRING  esto para provocar una excepción, entonces las instrucciones INSERT de las lineas **30,35,40** se ejecutaran pero no así la linea **47**.





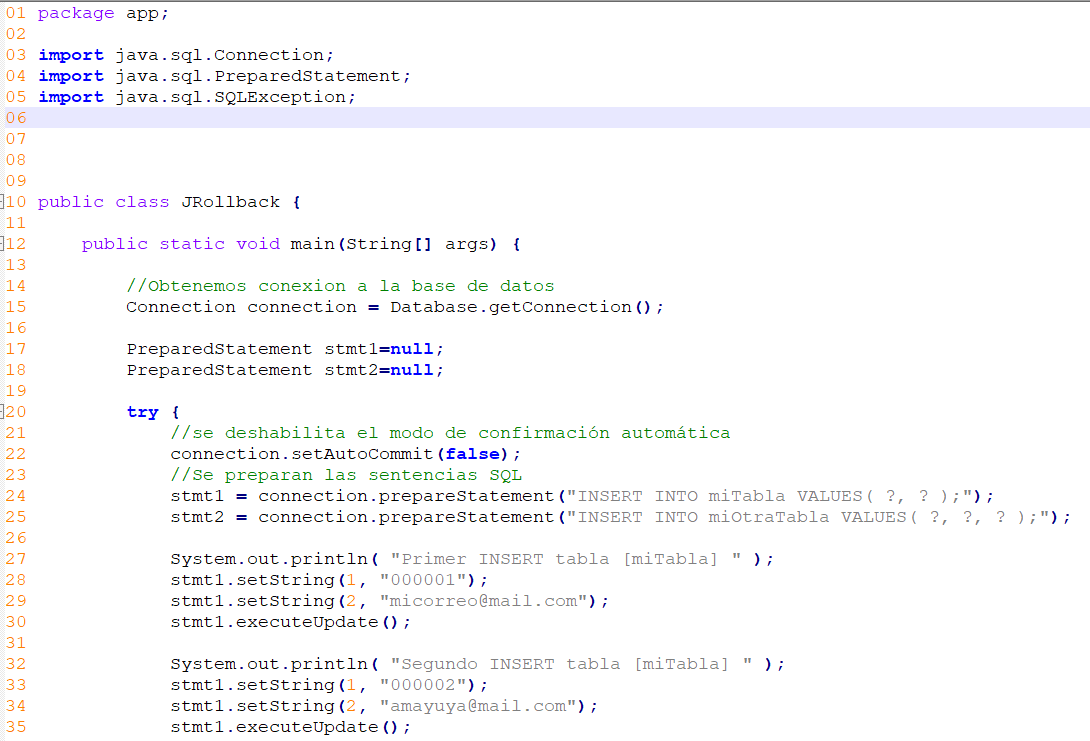
Ejecutando la clase Main *JNoRollback* obtenemos el siguiente resultado por consola:

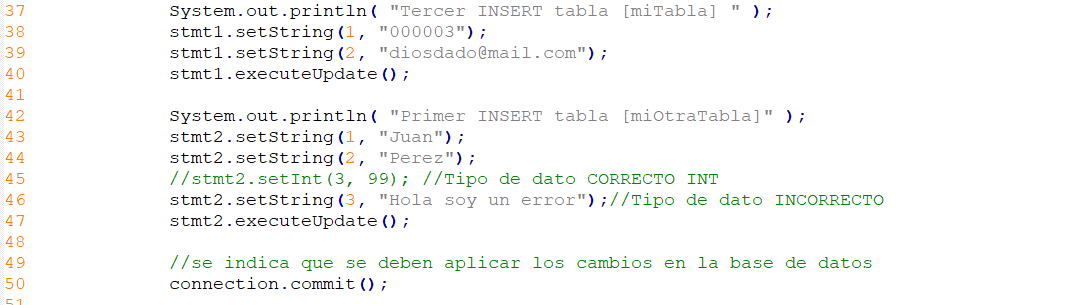


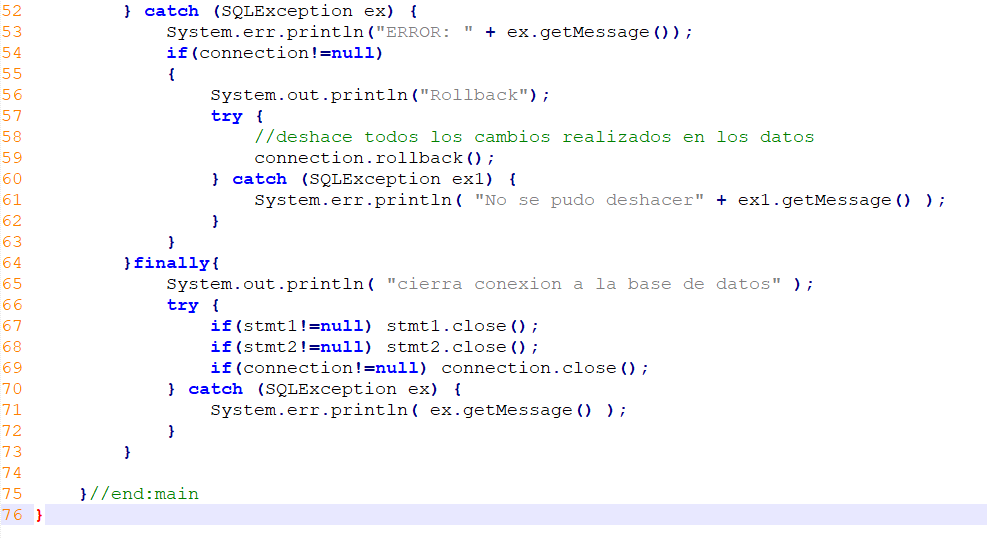
Como se mencionó líneas arriba, al ejecutar este código se produce una excepción «**ERROR: Incorrect integer value: ‘Hola soy un error’ for column ‘edad’ at row 1**» y si revisamos la base de datos, podremos observar como en la tabla «**miTabla**» se registraron 3 nuevas filas pero si abrimos la tabla «**miOtraTabla**» no existen registros, es decir no se está cumpliendo con lo principio de ***ATOMICIDAD Y CONSISTENCIA***

**Clase JRollback.java**

Esta clase contiene el mismo código de más arriba pero se le agrego algunas líneas de código extra







**Explicación:**

**Explicación:**

**22** : Se deshabilita el modo de confirmación automática, es decir todas las instrucciones SQL a partir de ese punto se agrupan en una sola transacción que finalizara con una llamada al método **commit** o al método **rollback**.

**45 :** Este es el tipo de dato correcto que acepta la instrucción SQL, para usarlo des-comenta y comenta la linea de código ***46***

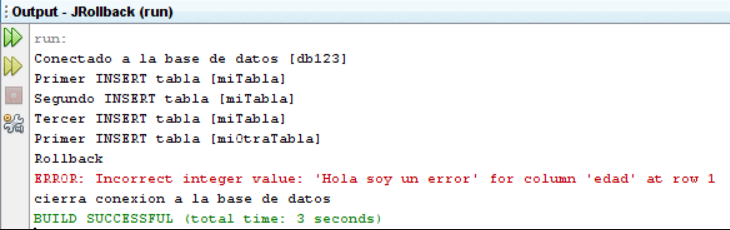
**46 :** El error de tipo de dato es el mismo que de la clase anterior

**50 :**  Aquí hacemos uso del método **commit()**, es decir si se llega hasta este punto quiere decir que no se encontró ningún error y por tanto todas las instrucciones SQL *(en este caso INSERT)* serán confirmadas.

**59 :** Si se produce alguna excepción, se hará un llamado al método **rollback()** este método deshace todos los cambios realizados en la transacción actual y libera cualquier bloque en la base de datos que contenga en esos momentos este objeto

**64 :** Para terminar se cierra la conexión actual a la base de datos

Ejecutando esta clase obtenemos la siguiente salida en pantalla:



Como en el caso anterior provocamos una excepción, sin embargo en esta oportunidad  la transacción no llega a ser confirmada (*commit*) y es anulada por el método *rollback*. Si nos fijamos en nuestra base de datos, veremos que tanto en la tabla «*miTabla*» y «*miOtraTabla*» no se registró nada.

**VII. RÚBRICA DE LA EVALUACIÓN**

|  |  |  |  |
| --- | --- | --- | --- |
| Inicio (M)  0-10 | Proceso (R)  11-13 | Logro previsto (R+)  14-17 | Logro satisfactorio (B)  18-20 |
| Desarrollo correctamente del laboratorio hasta un 50 % | Desarrollo correctamente del laboratorio hasta un 60 % | Desarrollo correctamente del laboratorio hasta un 80 % | Desarrollo correctamente del laboratorio hasta un 100% |