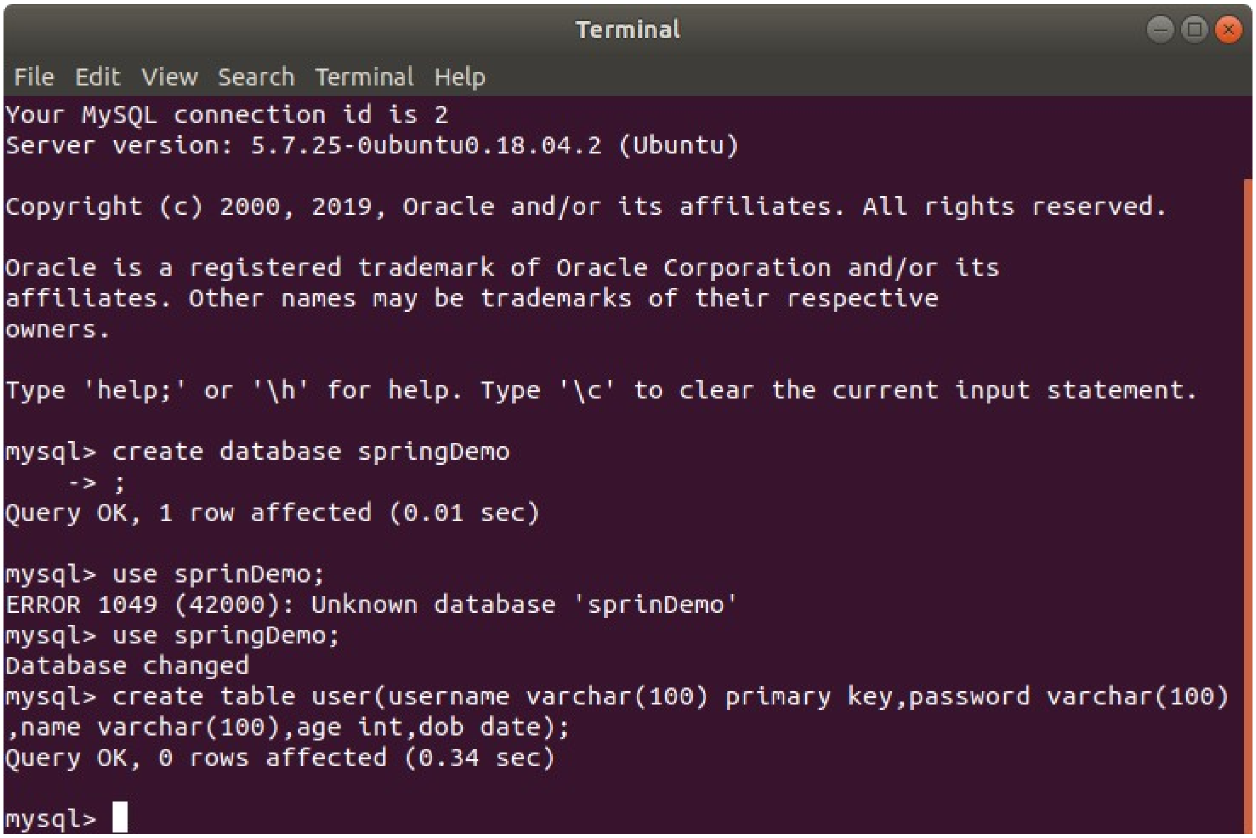
Q1. Create database springDemo with user table with fields

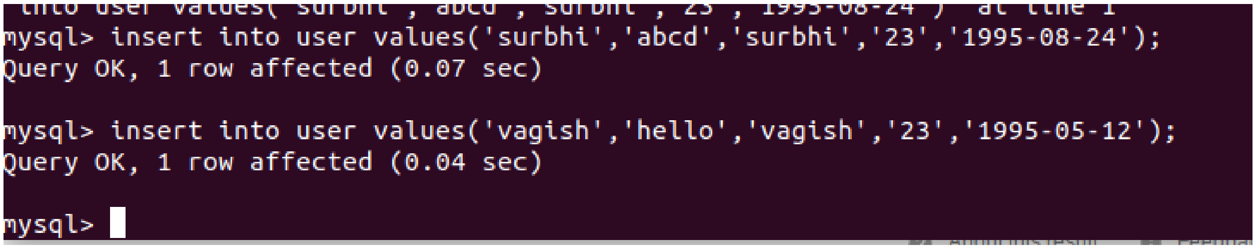
(a) username
(b) password
(c) name
(d) age
(e) dob

Solution



```
Your MySQL connection id is 2
Server version: 5.7.25-0ubuntu0.18.04.2 (Ubuntu)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database springDemo
    -> ;
Query OK, 1 row affected (0.01 sec)

mysql> use sprinDemo;
ERROR 1049 (42000): Unknown database 'sprinDemo'
mysql> use springDemo;
Database changed
mysql> create table user(username varchar(100) primary key,password varchar(100)
,name varchar(100),age int,dob date);
Query OK, 0 rows affected (0.34 sec)

mysql> 
```

Q2. Insert few records inside user Tables



```
 into user values('surbhi','abcd','surbhi','23','1995-08-24')' at line 1
mysql> insert into user values('surbhi','abcd','surbhi','23','1995-08-24');
Query OK, 1 row affected (0.07 sec)

mysql> insert into user values('vagish','hello','vagish','23','1995-05-12');
Query OK, 1 row affected (0.04 sec)

mysql> 
```

Q3. Use datasource with DriverManagerDataSource, dbcp2.BasicDataSource and Q4. SingleConnectionDataSource to print the records of user tables

Solution

Build.gradle

```
dependencies
{
        compile group: 'org.springframework', name: 'spring-jdbc', version: '5.0.2.RELEASE'
        compile group: 'org.apache.commons', name: 'commons-dbcp2', version: '2.1'
}
```

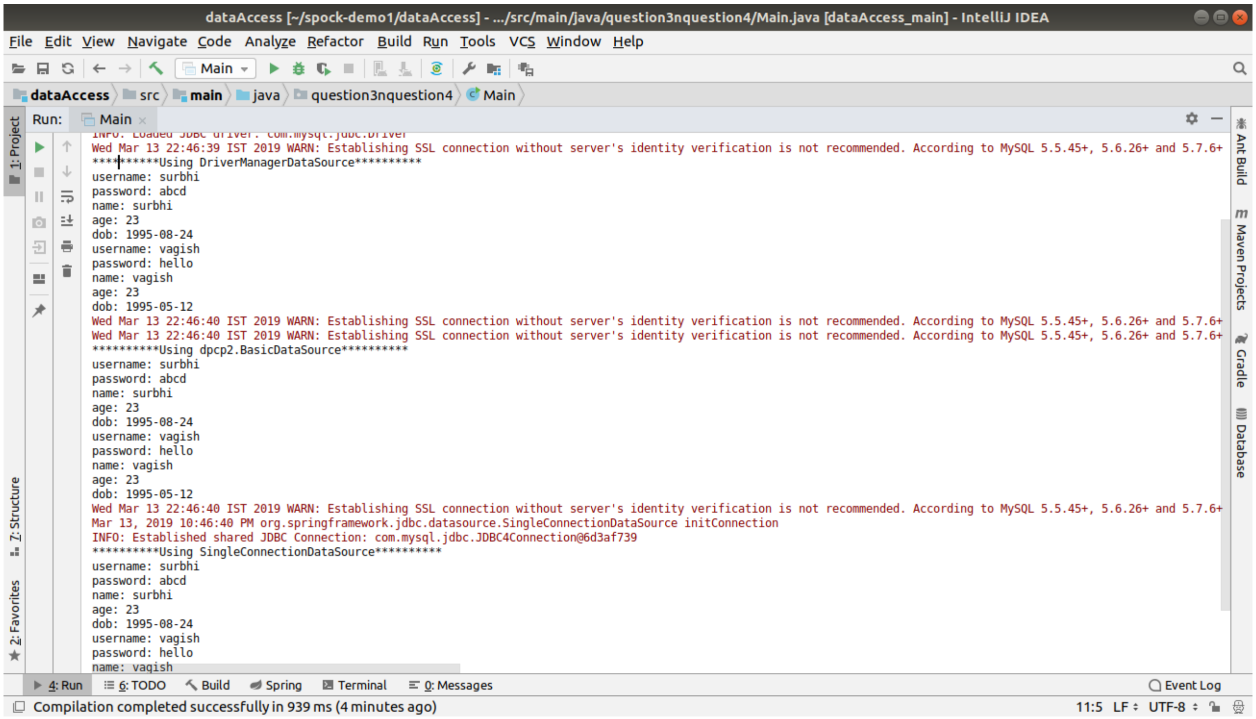
Spring-config.xml

```
<!--Question3-DriverManagerDataSource-->
<bean name="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="url" value="jdbc:mysql://localhost:3306/springDemo"/>
  <property name="username" value="root"/>
  <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
  <property name="password" value="root@123"/>
</bean>

<!--Question3-dpcp2.BasicDataSource-->
<bean name="basicDataSource" class="org.apache.commons.dbcp2.BasicDataSource">
  <property name="url" value="jdbc:mysql://localhost:3306/springDemo"/>
  <property name="username" value="root"/>
  <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
  <property name="password" value="root@123"/>
</bean>

<!--Question4-SingleConnectionDataSource-->
<bean name="singleConnectionDataSource"
class="org.springframework.jdbc.datasource.SingleConnectionDataSource">
  <property name="url" value="jdbc:mysql://localhost:3306/springDemo"/>
  <property name="username" value="root"/>
  <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
  <property name="password" value="root@123"/>
</bean>
```

Main.java

```
package demo.question3nquestion4;

import org.apache.commons.dbcp2.BasicDataSource;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.datasource.SingleConnectionDataSource;

import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class Main {
  static void printUserResultSet(ResultSet set) throws SQLException {
        while(set.next())
        {
            System.out.println("username: "+set.getString("username"));
            System.out.println("password: "+set.getString("password"));
            System.out.println("name: "+set.getString("name"));
            System.out.println("age: "+set.getInt("age"));
            System.out.println("dob: "+set.getDate("dob"));
        }
    }
    public static void main(String[] args) throws SQLException {
        ApplicationContext context=new ClassPathXmlApplicationContext("spring-config.xml");
        final String query="select * from user";

        //DriverManagerDataSource
        DataSource dataSource=context.getBean("dataSource",DataSource.class);
        Connection connection=dataSource.getConnection();

        PreparedStatement preparedStatement=connection.prepareStatement(query);
        ResultSet resultSet=preparedStatement.executeQuery();

        System.out.println("**********Using DriverManagerDataSource**********");
        printUserResultSet(resultSet);

        //dbcp2.BasicDataSource
        BasicDataSource basicDataSource=context.getBean(BasicDataSource.class);
        Connection basicDataSourceConnection=basicDataSource.getConnection();

        PreparedStatement preparedStatement1=basicDataSourceConnection.prepareStatement(query);
        resultSet=preparedStatement1.executeQuery();

        System.out.println("**********Using dpcp2.BasicDataSource**********");
        printUserResultSet(resultSet);

        //SingleDataSource
        SingleConnectionDataSource
singleConnectionDataSource=context.getBean(SingleConnectionDataSource.class);
        Connection connection1=singleConnectionDataSource.getConnection();

        PreparedStatement preparedStatement2=connection1.prepareStatement(query);
        resultSet=preparedStatement2.executeQuery();

        System.out.println("**********Using SingleConnectionDataSource**********");
        printUserResultSet(resultSet);


        //




    }
}
```

Output



dataAccess [~/spock-demo1/dataAccess] - .../src/main/java/question3nquestion4/Main.java [dataAccess_main] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Main
dataAccess
src
main
java
question3nquestion4
Main
Run:
Main
1: Project
7: Structure
2: Favorites
Ant Build
Maven Projects
Gradle
Database
4: Run
6: TODO
Build
Spring
Terminal
0: Messages
Event Log
11:5
LF
UTF-8

```
INFO: Loaded JDBC driver: com.mysql.jdbc.Driver
Wed Mar 13 22:46:39 IST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+
**********Using DriverManagerDataSource**********
username: surbhi
password: abcd
name: surbhi
age: 23
dob: 1995-08-24
username: vagish
password: hello
name: vagish
age: 23
dob: 1995-05-12
Wed Mar 13 22:46:40 IST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+
Wed Mar 13 22:46:40 IST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+
**********Using dpcp2.BasicDataSource**********
username: surbhi
password: abcd
name: surbhi
age: 23
dob: 1995-08-24
username: vagish
password: hello
name: vagish
age: 23
dob: 1995-05-12
Wed Mar 13 22:46:40 IST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+
Mar 13, 2019 10:46:40 PM org.springframework.jdbc.datasource.SingleConnectionDataSource initConnection
INFO: Established shared JDBC Connection: com.mysql.jdbc.JDBC4Connection@6d3af739
**********Using SingleConnectionDataSource**********
username: surbhi
password: abcd
name: surbhi
age: 23
dob: 1995-08-24
username: vagish
password: hello
name: vagish
```

Compilation completed successfully in 939 ms (4 minutes ago)

Q5. Use JdbcTemplate to get the count of users

Solution

Spring-config.xml

```
<!--Question5 to11-->
<bean class="org.springframework.jdbc.core.JdbcTemplate">
  <property name="dataSource" ref="dataSource"/>
</bean>
<context:component-scan base-package="demo.*"/>
```

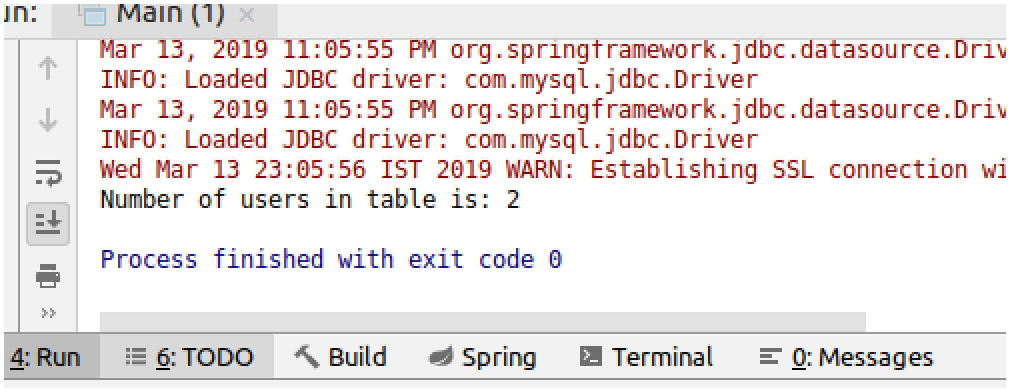
UserDao.java

```
package demo.question5to11;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import javax.sql.DataSource;

@Repository
public class UserDao {
  @Autowired
  private DataSource dataSource;
  @Autowired
  private JdbcTemplate jdbcTemplate;

 //Question5
  public int countAllUsers()
  {
     final String query="select count(*) from user";
      int count= jdbcTemplate.queryForObject(query,Integer.class);
      return count;
  }
}
```

Main.java

```
package demo.question5to11;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {
  public static void main(String[] args) {
     ApplicationContext applicationContext=new ClassPathXmlApplicationContext("spring-config.xml");
     UserDao userDao=applicationContext.getBean(UserDao.class);
     //Question5
     System.out.println(“Number of users in table is: "+userDao.countAllUsers());
  }
}
```

Output



```
Mar 13, 2019 11:05:55 PM org.springframework.jdbc.datasource.Driv
INFO: Loaded JDBC driver: com.mysql.jdbc.Driver
Mar 13, 2019 11:05:55 PM org.springframework.jdbc.datasource.Driv
INFO: Loaded JDBC driver: com.mysql.jdbc.Driver
Wed Mar 13 23:05:56 IST 2019 WARN: Establishing SSL connection wi
Number of users in table is: 2

Process finished with exit code 0
```

Q6. Get name of the user by providing username to the parametrized query

Solution

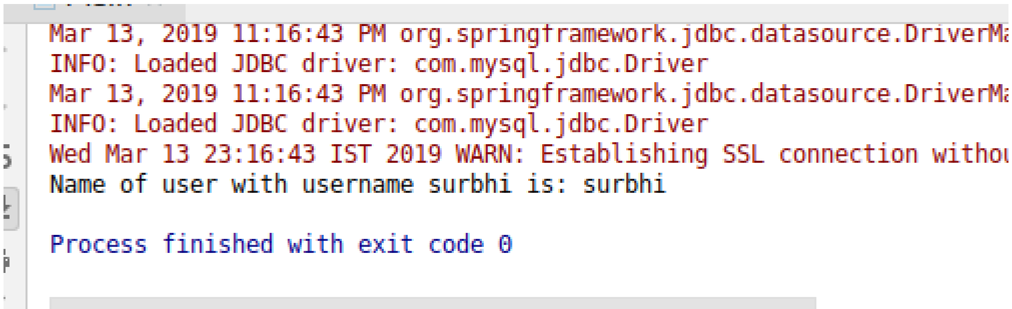
UserDao.java

```
package demo.question5to11;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

@Repository
public class UserDao {
  @Autowired
  private JdbcTemplate jdbcTemplate;

  //Question6
  public String findNameByUserName(String username)
  {
     final String query="select name from user where username=?";
     String name=jdbcTemplate.queryForObject(query,new Object[]{username},String.class);
     return name;
  }
}
```

Main.java

```
package demo.question5to11;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {
  public static void main(String[] args) {
     ApplicationContext applicationContext=new ClassPathXmlApplicationContext("spring-config.xml");
     UserDao userDao=applicationContext.getBean(UserDao.class);
     //Question6
     System.out.println("Name of user with username surbhi is: "+userDao.findNameByUserName("surbhi"));
  }
}
```

Output



```
Mar 13, 2019 11:16:43 PM org.springframework.jdbc.datasource.DriverMa
INFO: Loaded JDBC driver: com.mysql.jdbc.Driver
Mar 13, 2019 11:16:43 PM org.springframework.jdbc.datasource.DriverMa
INFO: Loaded JDBC driver: com.mysql.jdbc.Driver
Wed Mar 13 23:16:43 IST 2019 WARN: Establishing SSL connection withou
Name of user with username surbhi is: surbhi

Process finished with exit code 0
```

Q7. Insert one record using JdbcTemplate

Solution

User.java

```java
package demo.question5to11;

import java.time.LocalDate;

public class Users {
  String userName;
  String name;
  String password;
  int age;
  LocalDate dob;

  public String getUserName() {
    return userName;
  }

  public void setUserName(String userName) {
    this.userName = userName;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getPassword() {
    return password;
  }

  public void setPassword(String password) {
    this.password = password;
  }

  public int getAge() {
    return age;
  }

  public void setAge(int age) {
    this.age = age;
  }

  public LocalDate getDob() {
    return dob;
  }

  public void setDob(LocalDate dob) {
    this.dob = dob;
  }
@Override
public String toString() {
  return "Users{" +
        "userName='" + userName + '\'' +
        ", name='" + name + '\'' +
        ", password='" + password + '\'' +
        ", age=" + age +
        ", dob=" + dob +
        '}';
}

}
```

UserDao.java

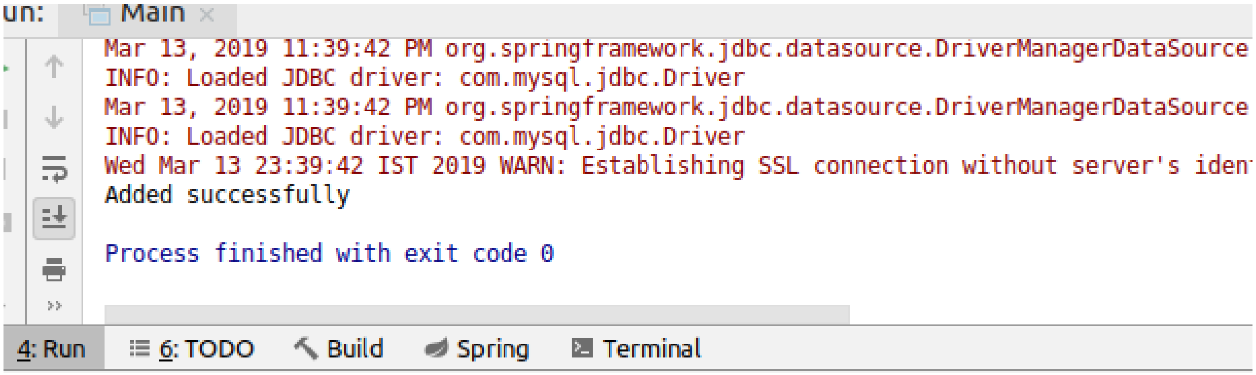
```
package demo.question5to11;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.sql.Date;

@Repository
public class UserDao {
  @Autowired
  private JdbcTemplate jdbcTemplate;
  //Question7
  public int addUser(Users user)
  {
     final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
    return jdbcTemplate.update(query,new Object[]
{user.getUserName(),user.getName(),user.getPassword(),user.getAge(),user.getDob()});
  }
}
```

Main.java

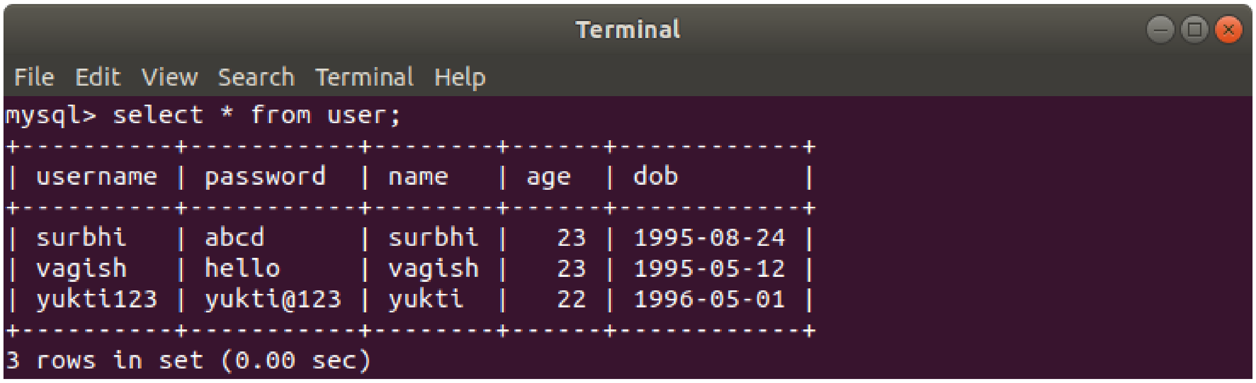
```
package demo.question5to11;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.sql.Date;
import java.time.LocalDate;

public class Main {
  public static void main(String[] args) {
     ApplicationContext applicationContext = new ClassPathXmlApplicationContext("spring-config.xml");
     UserDao userDao = applicationContext.getBean(UserDao.class);

     //Question7
     Users user=new Users();
     user.setName("yukti");
     user.setUserName("yukti123");
```

```
        user.setAge(22);
        user.setPassword("yukti@123");
        user.setDob(LocalDate.parse("1996-05-01"));

        if(userDao.addUser(user)>0)
        System.out.println("Added successfully");
        else
        System.out.println("Some error occurred");

    }
}
```

Output



```
Mar 13, 2019 11:39:42 PM org.springframework.jdbc.datasource.DriverManagerDataSource
INFO: Loaded JDBC driver: com.mysql.jdbc.Driver
Mar 13, 2019 11:39:42 PM org.springframework.jdbc.datasource.DriverManagerDataSource
INFO: Loaded JDBC driver: com.mysql.jdbc.Driver
Wed Mar 13 23:39:42 IST 2019 WARN: Establishing SSL connection without server's iden
Added successfully

Process finished with exit code 0
```



```
mysql> select * from user;
+----------+-----------+--------+------+------------+
| username | password  | name   | age  | dob        |
+----------+-----------+--------+------+------------+
| surbhi   | abcd      | surbhi |   23 | 1995-08-24 |
| vagish   | hello     | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+-----------+--------+------+------------+
3 rows in set (0.00 sec)
```

Q8. Use QueryForMap to fetch the user details of the user

Solution

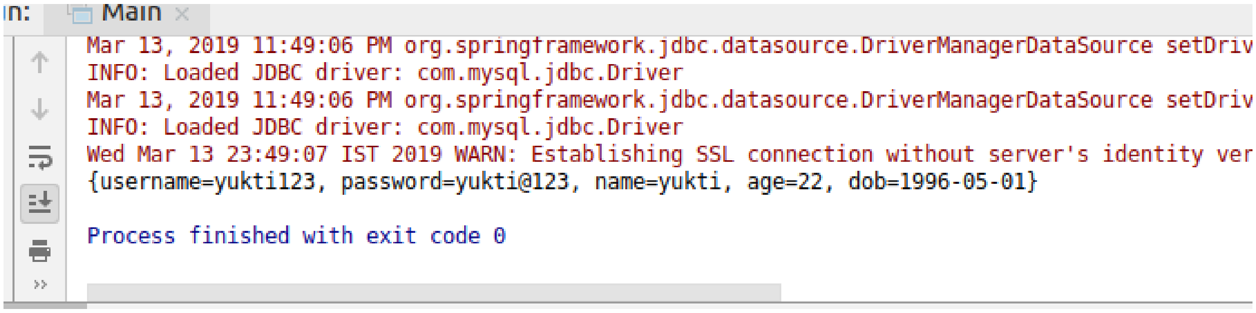
UserDao.java

```
package demo.question5to11;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.sql.Date;

@Repository
public class UserDao {
  @Autowired
  private JdbcTemplate jdbcTemplate;
  //Question8
  public void findUserByUserName(String username)
  {
    final String query="select * from user where username=?";
    System.out.println(jdbcTemplate.queryForMap(query,new Object[]{username}));
  }
}
```

Main.java

```
package demo.question5to11;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.sql.Date;
import java.time.LocalDate;

public class Main {
  public static void main(String[] args) {
    ApplicationContext applicationContext = new ClassPathXmlApplicationContext("spring-config.xml");
    UserDao userDao = applicationContext.getBean(UserDao.class);
    //Question8
    userDao.findUserByUserName("yukti123");

  }
}
```

Output



```
Mar 13, 2019 11:49:06 PM org.springframework.jdbc.datasource.DriverManagerDataSource setDriv
INFO: Loaded JDBC driver: com.mysql.jdbc.Driver
Mar 13, 2019 11:49:06 PM org.springframework.jdbc.datasource.DriverManagerDataSource setDriv
INFO: Loaded JDBC driver: com.mysql.jdbc.Driver
Wed Mar 13 23:49:07 IST 2019 WARN: Establishing SSL connection without server's identity ver
{username=yukti123, password=yukti@123, name=yukti, age=22, dob=1996-05-01}

Process finished with exit code 0
```

Q9. Use QueryForList to fetch records of all users

Solution

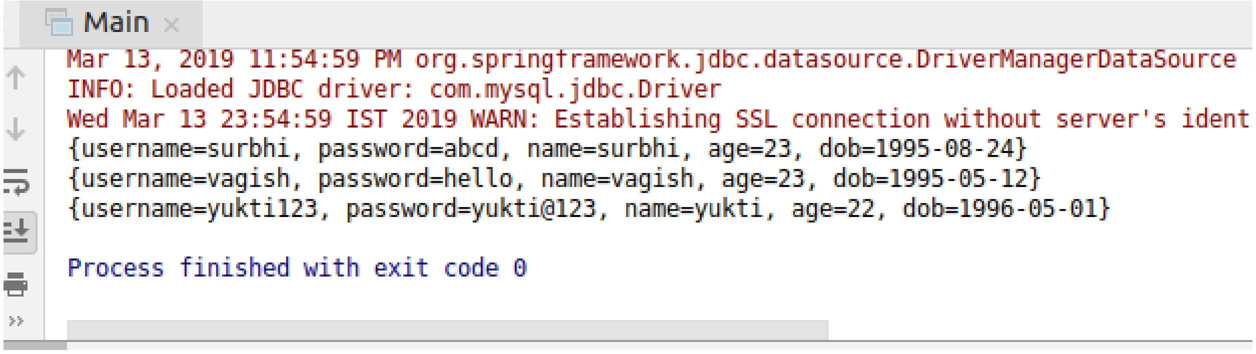
UserDao.java

```
package demo.question5to11;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.sql.Date;

@Repository
public class UserDao {
  @Autowired
  private JdbcTemplate jdbcTemplate;
  //Question9
  public void printAllUsers()
  {
     final String query="select * from user";
     jdbcTemplate.queryForList(query).forEach(System.out::println);
  }
}
```

Main.java

```
package demo.question5to11;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.sql.Date;
import java.time.LocalDate;

public class Main {
  public static void main(String[] args) {
     ApplicationContext applicationContext = new ClassPathXmlApplicationContext("spring-config.xml");
     UserDao userDao = applicationContext.getBean(UserDao.class);
     //Question9
     userDao.printAllUsers();

  }
}
```

Output



```
Mar 13, 2019 11:54:59 PM org.springframework.jdbc.datasource.DriverManagerDataSource
INFO: Loaded JDBC driver: com.mysql.jdbc.Driver
Wed Mar 13 23:54:59 IST 2019 WARN: Establishing SSL connection without server's ident
{username=surbhi, password=abcd, name=surbhi, age=23, dob=1995-08-24}
{username=vagish, password=hello, name=vagish, age=23, dob=1995-05-12}
{username=yukti123, password=yukti@123, name=yukti, age=22, dob=1996-05-01}

Process finished with exit code 0
```

Q10. Use a rowmapper to get the User object when you query for a user

Solution

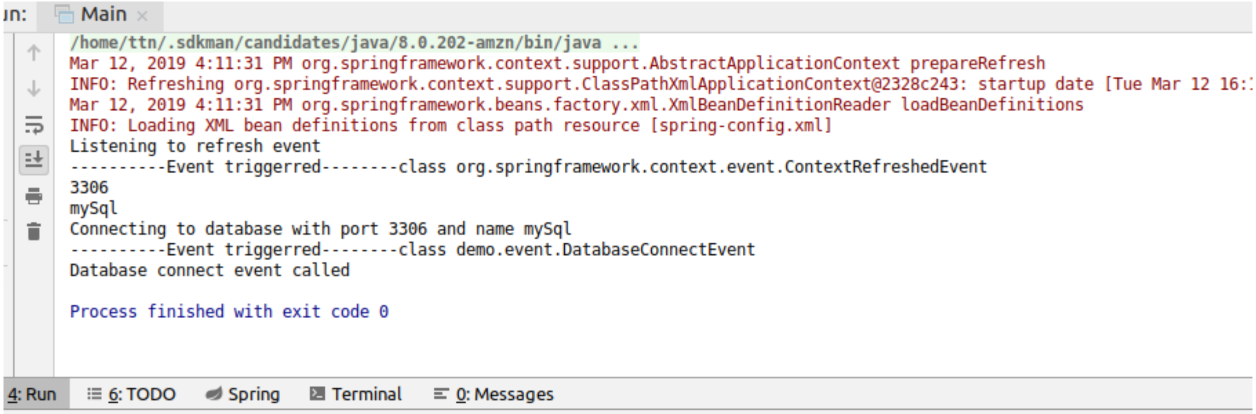
UserDao.java

```
package demo.question5to11;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Repository;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDate;

@Repository
public class UserDao {
  @Autowired
  private JdbcTemplate jdbcTemplate;
//Question10
Users getUser(String username)
{
  final String query="select * from user where username=?";
  Users users=jdbcTemplate.queryForObject(query, new Object[]{username},(rs,rowNum)->
      {

          Users user=new Users();
          user.setUserName(rs.getString("username"));
          user.setName(rs.getString("name"));
          user.setPassword(rs.getString("password"));
          user.setAge(rs.getInt("age"));
          user.setDob(rs.getDate("dob").toLocalDate());
          return user;
      });
  return users;
}

}
```

Main.java

```
package demo.question5to11;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.time.LocalDate;

public class Main {
  public static void main(String[] args) {
      ApplicationContext applicationContext = new ClassPathXmlApplicationContext("spring-config.xml");
      UserDao userDao = applicationContext.getBean(UserDao.class);
      //Question10
      Users user=userDao.getUser("surbhi");
      System.out.println(user);
  }
}
```

Output



```
/home/ttn/.sdkman/candidates/java/8.0.202-amzn/bin/java ...
Mar 12, 2019 4:11:31 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@2328c243: startup date [Tue Mar 12 16:
Mar 12, 2019 4:11:31 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [spring-config.xml]
Listening to refresh event
----------Event triggerred--------class org.springframework.context.event.ContextRefreshedEvent
3306
mySql
Connecting to database with port 3306 and name mySql
----------Event triggerred--------class demo.event.DatabaseConnectEvent
Database connect event called

Process finished with exit code 0
```

Q11. Integrate Hibernate with Spring and use hql query to count the number of records in user table.

Solution

Build.gradle

```
dependencies
{
compile group: 'org.springframework', name: 'spring-orm', version: '4.0.3.RELEASE'
compile group: 'org.hibernate', name: 'hibernate-core', version: '5.4.1.Final'
}
```

Spring-config.xml

```
<!--Question11-->
<bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
  <property name="dataSource" ref="dataSource"/>
  <property name="packagesToScan" value="demo.*"/>
  <property name="hibernateProperties">
    <props>
      <prop key="dialect">org.hibernate.dialect.MySQLDialect</prop>
    </props>
  </property>
</bean>
```

User.java

```
package demo.question5to11;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import java.io.Serializable;
import java.time.LocalDate;

//added annotations for question11
@Entity
@Table(name = "user")
public class Users implements Serializable {
  @Id
  @Column(name = "username")
  String userName;
  @Column(name = "name")
  String name;
  @Column(name = "password")
  String password;
  @Column(name="age")
  int age;
  @Column(name = "dob")
  LocalDate dob;

  public String getUserName() {
    return userName;
  }

  public void setUserName(String userName) {
```

```java
        this.userName = userName;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public LocalDate getDob() {
        return dob;
    }

    public void setDob(LocalDate dob) {
        this.dob = dob;
    }

    @Override
    public String toString() {
        return "Users{" +
                "userName='" + userName + '\'' +
                ", name='" + name + '\'' +
                ", password='" + password + '\'' +
                ", age=" + age +
                ", dob=" + dob +
                '}';
    }
}
```

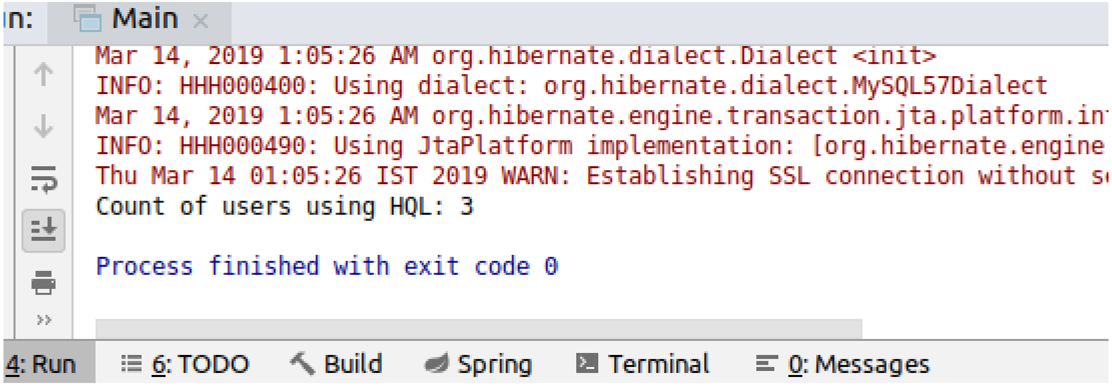
UserDao.java

```java
package demo.question5to11;

import org.hibernate.SessionFactory;
import org.hibernate.query.Query;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

@Repository
public class UserDao {
  @Autowired
  private JdbcTemplate jdbcTemplate;

  //forQuestion11
  @Autowired
  SessionFactory sessionFactory;

  //Question11
  public void countUsersUsingHql()
  {
    String queryString="select count(distinct u.userName) from Users u";
    Query query=sessionFactory.openSession().createQuery(queryString);
    System.out.println("Count of users using HQL: "+query.uniqueResult());
  }
}
```

Main.java

```
package demo.question5to11;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {
  public static void main(String[] args) {
    ApplicationContext applicationContext = new ClassPathXmlApplicationContext("spring-config.xml");
    UserDao userDao = applicationContext.getBean(UserDao.class);
//      Question11
    userDao.countUsersUsingHql();
  }
}
```

Output



```
Mar 14, 2019 1:05:26 AM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL57Dialect
Mar 14, 2019 1:05:26 AM org.hibernate.engine.transaction.jta.platform.in
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine
Thu Mar 14 01:05:26 IST 2019 WARN: Establishing SSL connection without s
Count of users using HQL: 3

Process finished with exit code 0
```

Q12. Use @Transactional to save to save 2 records using jdbc template with the following prapogation options

- REQUIRED
- REQUIRES_NEW
- NESTED
- MANDATORY
- NEVER
- NOT_SUPPORTED
- SUPPORTS

Solution

REQUIRED

UserDao.java

```
package demo.question12.required;

import demo.question5to11.Users;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.stereotype.Repository;

import org.springframework.transaction.annotation.Transactional;

@Repository("userdao")

public class UserDao {

  @Autowired

  JdbcTemplate jdbcTemplate;

  @Autowired

  UserDao2 userDao2;

  @Transactional

  public void addTwoUsers(Users user1,Users user2)

  {

    final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";

    jdbcTemplate.update(query,new Object[]{user1.getUserName(), user1.getName(), user1.getPassword(), user1.getAge(), user1.getDob()});

          try {

              userDao2.addUser(user2);
```

```
        }
        catch (Exception ex)
        {
          ex.printStackTrace();
        }



  }
}
```

UserDao2.java

```
package demo.question12.required;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Required;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;


@Repository
public class UserDao2 {
  @Autowired
  JdbcTemplate jdbcTemplate;
  @Transactional(propagation=Propagation.REQUIRED)
  public void addUser(Users user) {
    final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
    jdbcTemplate.update(query,new Object[]{user.getUserName(), user.getName(), user.getPassword(),
user.getAge(), user.getDob()});
//        throw new RuntimeException();
  }
}
```

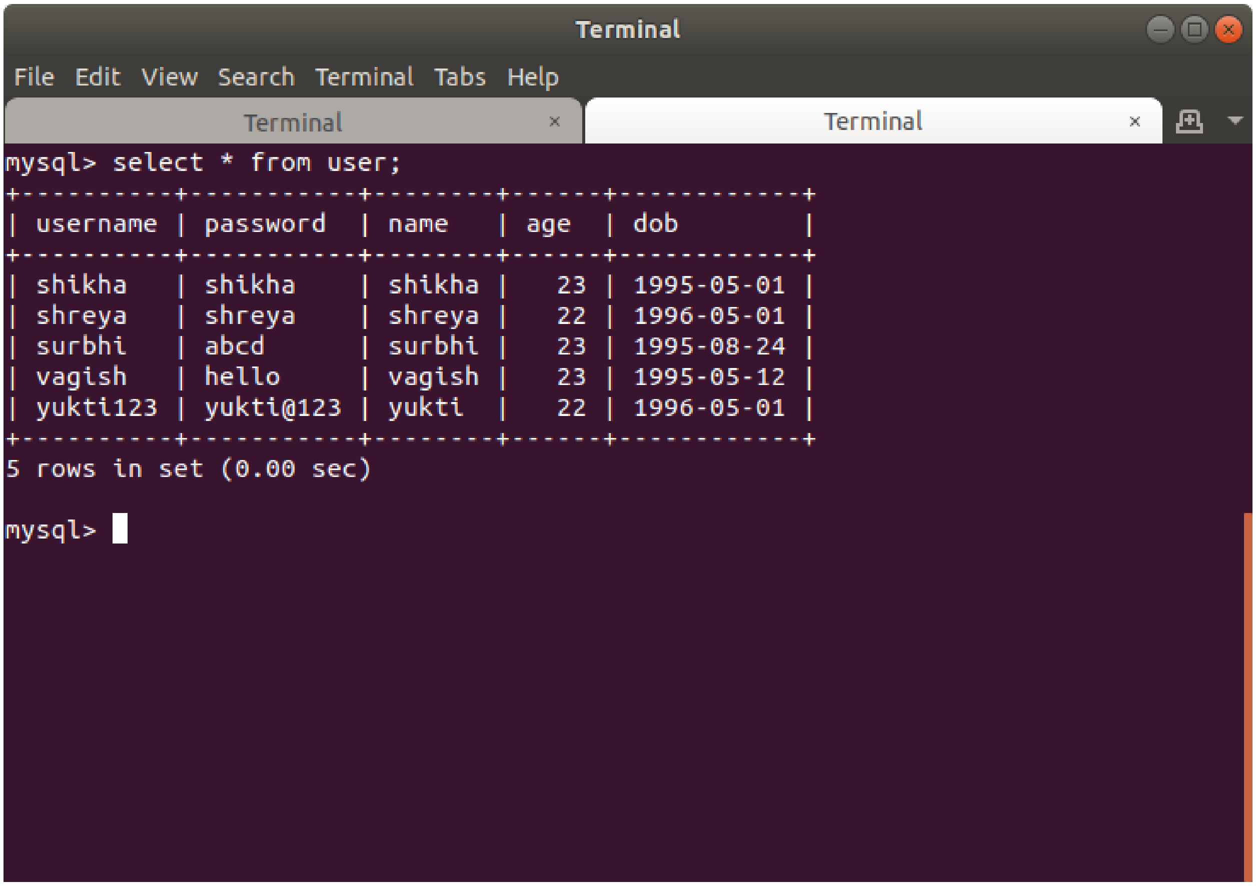
Main.java

```
package demo.question12.required;

import demo.question5to11.Users;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.time.LocalDate;

public class Main {
  public static void main(String[] args) {
     ApplicationContext applicationContext=new ClassPathXmlApplicationContext("spring-config.xml");
    UserDao userDao= applicationContext.getBean(UserDao.class);
     Users user=new Users();
     user.setUserName("shreya");
     user.setName("shreya");
     user.setPassword("shreya");
     user.setAge(22);
     user.setDob(LocalDate.parse("1996-05-01"));

     Users user2=new Users();
     user2.setUserName("shikha");
     user2.setName("shikha");
     user2.setPassword("shikha");
     user2.setAge(23);
```

```
        user2.setDob(LocalDate.parse("1995-05-01"));


        userDao.addTwoUsers(user,user2);
    }
}
```

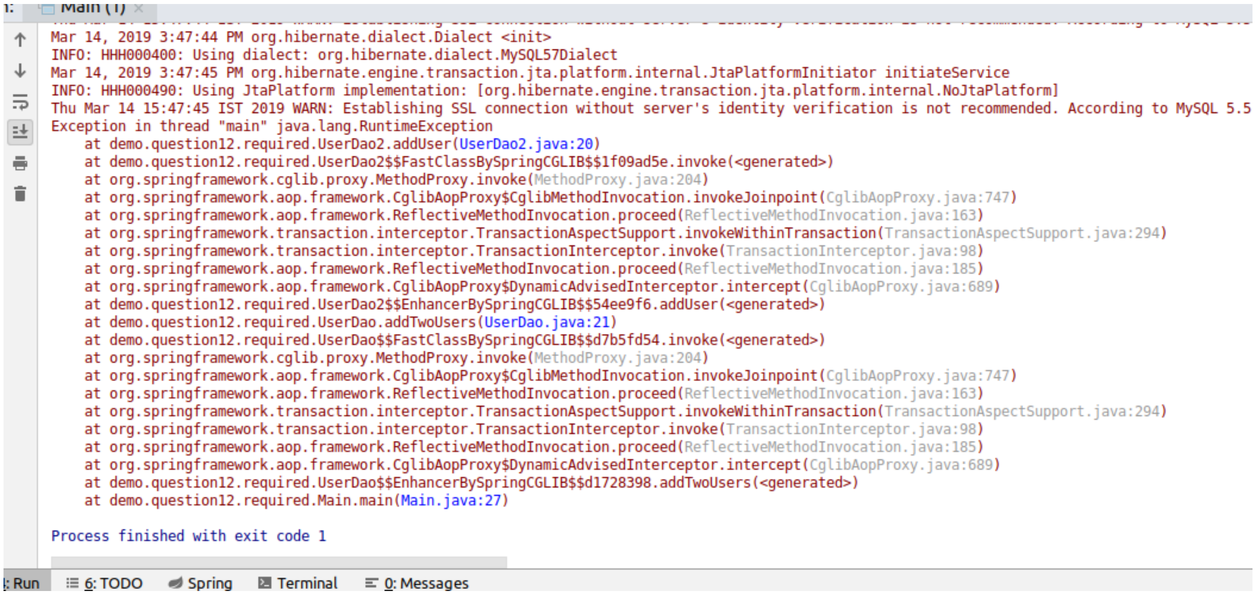
Inference

Both users get added into user table



```
mysql> select * from user;
+----------+-----------+--------+------+------------+
| username | password  | name   | age  | dob        |
+----------+-----------+--------+------+------------+
| shikha   | shikha    | shikha |   23 | 1995-05-01 |
| shreya   | shreya    | shreya |   22 | 1996-05-01 |
| surbhi   | abcd      | surbhi |   23 | 1995-08-24 |
| vagish   | hello     | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+-----------+--------+------+------------+
5 rows in set (0.00 sec)

mysql> 
```

If i delete both the users added, and uncomment throw new RunTimeException() in UserDao2.java and run main, no user will be saved, and an exception will be thrown



```
Mar 14, 2019 3:47:44 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL57Dialect
Mar 14, 2019 3:47:45 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
Thu Mar 14 15:47:45 IST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5
Exception in thread "main" java.lang.RuntimeException
    at demo.question12.required.UserDao2.addUser(UserDao2.java:20)
    at demo.question12.required.UserDao2$$FastClassBySpringCGLIB$$1f09ad5e.invoke(<generated>)
    at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:204)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibAopProxy.java:747)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163)
    at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:294)
    at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:98)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:185)
    at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:689)
    at demo.question12.required.UserDao2$$EnhancerBySpringCGLIB$$54ee9f6.addUser(<generated>)
    at demo.question12.required.UserDao.addTwoUsers(UserDao.java:21)
    at demo.question12.required.UserDao$$FastClassBySpringCGLIB$$d7b5fd54.invoke(<generated>)
    at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:204)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibAopProxy.java:747)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163)
    at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:294)
    at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:98)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:185)
    at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:689)
    at demo.question12.required.UserDao$$EnhancerBySpringCGLIB$$d1728398.addTwoUsers(<generated>)
    at demo.question12.required.Main.main(Main.java:27)

Process finished with exit code 1
```

REQUIRES_NEW

UserDao.java

```java
package demo.question12.requiresnew;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdaorequiresnew")
public class UserDao {
   @Autowired
   JdbcTemplate jdbcTemplate;
   @Autowired
   UserDao2 userDao2;
   @Transactional
   public void addTwoUsers(Users user1,Users user2)
   {
       final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
       jdbcTemplate.update(query,new Object[]{user1.getUserName(), user1.getName(), user1.getPassword(), user1.getAge(), user1.getDob()});
       try {
           userDao2.addUser(user2);
       }
       catch (Exception ex)
       {
           ex.printStackTrace();
       }


   }

}
```

UserDao2.java

```java
package demo.question12.requiresnew;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdao2requiresnew")
public class UserDao2 {
   @Autowired
   JdbcTemplate jdbcTemplate;
```

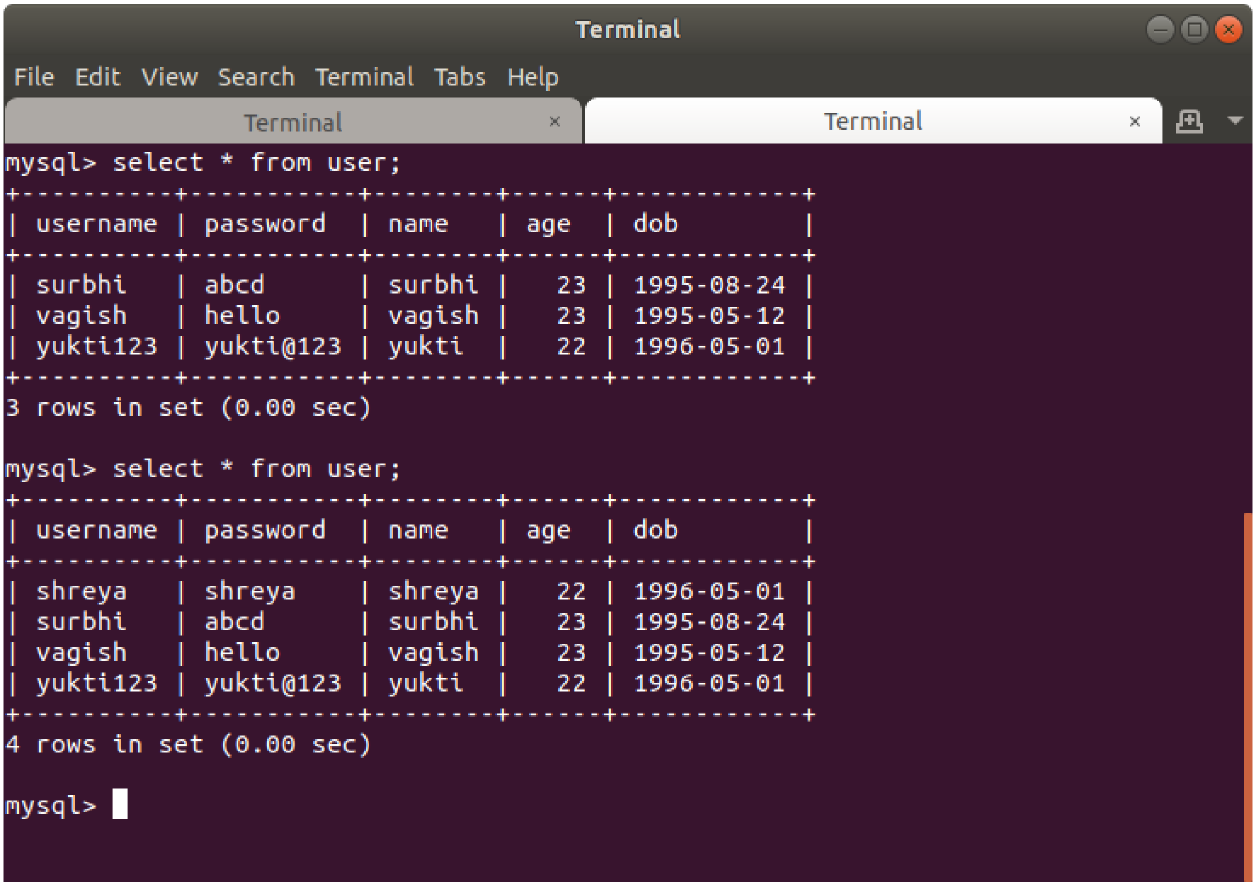
```
  @Transactional(propagation=Propagation.REQUIRES_NEW)
  public void addUser(Users user) {
    final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
    jdbcTemplate.update(query,new Object[]{user.getUserName(), user.getName(), user.getPassword(),
user.getAge(), user.getDob()});
    throw new RuntimeException();
  }
}
```

Main.java

```
package demo.question12.requiresnew;

import demo.question5to11.Users;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.time.LocalDate;

public class Main {
  public static void main(String[] args) {
    ApplicationContext applicationContext=new ClassPathXmlApplicationContext("spring-config.xml");
    UserDao userDao= applicationContext.getBean(UserDao.class);
    Users user=new Users();
    user.setUserName("shreya");
    user.setName("shreya");
    user.setPassword("shreya");
    user.setAge(22);
    user.setDob(LocalDate.parse("1996-05-01"));

    Users user2=new Users();
    user2.setUserName("shikha");
    user2.setName("shikha");
    user2.setPassword("shikha");
    user2.setAge(23);
    user2.setDob(LocalDate.parse("1995-05-01"));

    userDao.addTwoUsers(user,user2);
  }
}
```

Inference

When exception in UserDao is thrown in this case, the rollback will happen only for the transaction started by user2,i.e., user “shreya” will be saved, but not “shikha” unlike required which didn’t saved any.



```
mysql> select * from user;
+----------+-----------+--------+------+------------+
| username | password  | name   | age  | dob        |
+----------+-----------+--------+------+------------+
| surbhi   | abcd      | surbhi |   23 | 1995-08-24 |
| vagish   | hello     | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+-----------+--------+------+------------+
3 rows in set (0.00 sec)

mysql> select * from user;
+----------+-----------+--------+------+------------+
| username | password  | name   | age  | dob        |
+----------+-----------+--------+------+------------+
| shreya   | shreya    | shreya |   22 | 1996-05-01 |
| surbhi   | abcd      | surbhi |   23 | 1995-08-24 |
| vagish   | hello     | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+-----------+--------+------+------------+
4 rows in set (0.00 sec)

mysql> 
```

NESTED

Works like a checkpoint, i.e, wherever we mark propagation=NESTED, any exception in that method will not rollback caller transactional methods, but unlike Requires_New it doesn't create a new transaction for that transactional method.

UserDao.java

```
package demo.question12.nested;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdaonested")
public class UserDao {
  @Autowired
  JdbcTemplate jdbcTemplate;
  @Autowired
  UserDao2 userDao2;
  @Transactional
  public void addTwoUsers(Users user1,Users user2)
  {
     final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
     jdbcTemplate.update(query,new Object[]{user1.getUserName(), user1.getName(), user1.getPassword(),
user1.getAge(), user1.getDob()});
     try {
        userDao2.addUser(user2);
     }
     catch (Exception ex)
     {
        ex.printStackTrace();
     }


  }

}
```

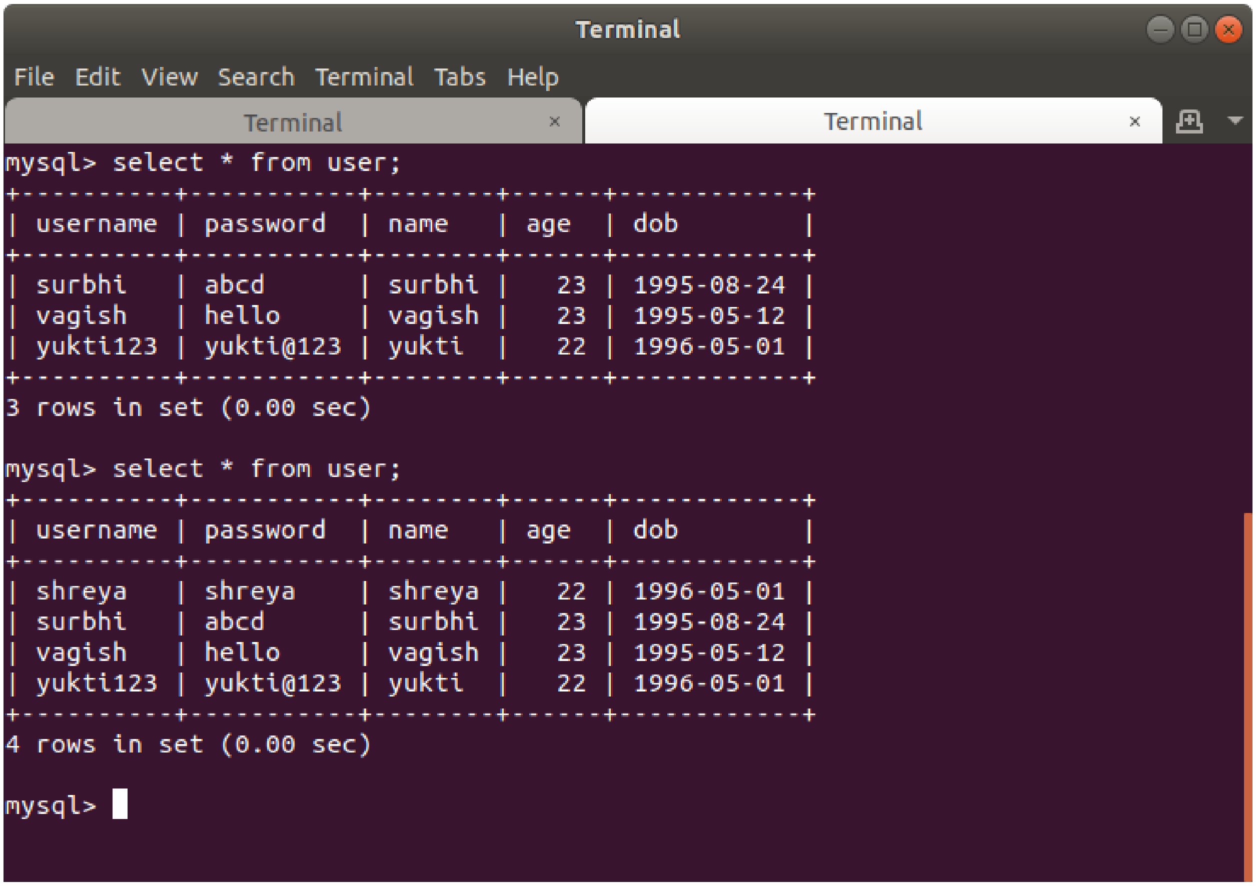
UserDao2.java

```
package demo.question12.nested;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdao2nested")
public class UserDao2 {
  @Autowired
```

```
    JdbcTemplate jdbcTemplate;
    @Transactional(propagation=Propagation.NESTED)
    public void addUser(Users user) {
        final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
        jdbcTemplate.update(query,new Object[]{user.getUserName(), user.getName(), user.getPassword(),
user.getAge(), user.getDob()});
        throw new RuntimeException();
    }
}
```

Main.java

```
package demo.question12.nested;

import demo.question5to11.Users;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.time.LocalDate;

public class Main {
    public static void main(String[] args) {
        ApplicationContext applicationContext=new ClassPathXmlApplicationContext("spring-config.xml");
       UserDao userDao= applicationContext.getBean(UserDao.class);
        Users user=new Users();
        user.setUserName("shreya");
        user.setName("shreya");
        user.setPassword("shreya");
        user.setAge(22);
        user.setDob(LocalDate.parse("1996-05-01"));

        Users user2=new Users();
        user2.setUserName("shikha");
        user2.setName("shikha");
        user2.setPassword("shikha");
        user2.setAge(23);
        user2.setDob(LocalDate.parse("1995-05-01"));

        userDao.addTwoUsers(user,user2);
    }
}
```



```
mysql> select * from user;
+----------+----------+--------+------+------------+
| username | password | name   | age  | dob        |
+----------+----------+--------+------+------------+
| surbhi   | abcd     | surbhi |   23 | 1995-08-24 |
| vagish   | hello    | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+----------+--------+------+------------+
3 rows in set (0.00 sec)

mysql> select * from user;
+----------+----------+--------+------+------------+
| username | password | name   | age  | dob        |
+----------+----------+--------+------+------------+
| shreya   | shreya   | shreya |   22 | 1996-05-01 |
| surbhi   | abcd     | surbhi |   23 | 1995-08-24 |
| vagish   | hello    | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+----------+--------+------+------------+
4 rows in set (0.00 sec)

mysql> 
```

MANDATORY

Requires that the caller of that transactional method should be transactional itself

UserDao.java

```
package demo.question12.mandatory;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdaomandatory")
public class UserDao {
   @Autowired
   JdbcTemplate jdbcTemplate;
   @Autowired
   UserDao2 userDao2;
  // @Transactional
   public void addTwoUsers(Users user1,Users user2)
   {
       final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
       jdbcTemplate.update(query,new Object[]{user1.getUserName(), user1.getName(), user1.getPassword(),
user1.getAge(), user1.getDob()});
       try {
           userDao2.addUser(user2);
       }
       catch (Exception ex)
       {
           ex.printStackTrace();
       }


   }

}
```

UserDao2.java

```
package demo.question12.mandatory;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdao2mandatory")
public class UserDao2 {
   @Autowired
   JdbcTemplate jdbcTemplate;
   @Transactional(propagation=Propagation.MANDATORY)
   public void addUser(Users user) {
```

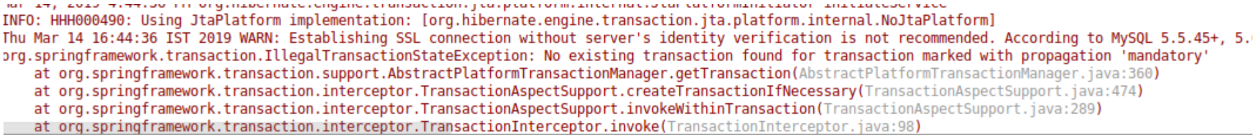
```java
        final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
        jdbcTemplate.update(query,new Object[]{user.getUserName(), user.getName(), user.getPassword(),
user.getAge(), user.getDob()});
        throw new RuntimeException();
    }
}
```

Main.java

```java
package demo.question12.mandatory;

import demo.question5to11.Users;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.time.LocalDate;

public class Main {
    public static void main(String[] args) {
        ApplicationContext applicationContext=new ClassPathXmlApplicationContext("spring-config.xml");
       UserDao userDao= applicationContext.getBean(UserDao.class);
        Users user=new Users();
        user.setUserName("shreya");
        user.setName("shreya");
        user.setPassword("shreya");
        user.setAge(22);
        user.setDob(LocalDate.parse("1996-05-01"));

        Users user2=new Users();
        user2.setUserName("shikha");
        user2.setName("shikha");
        user2.setPassword("shikha");
        user2.setAge(23);
        user2.setDob(LocalDate.parse("1995-05-01"));

        userDao.addTwoUsers(user,user2);
    }
}
```

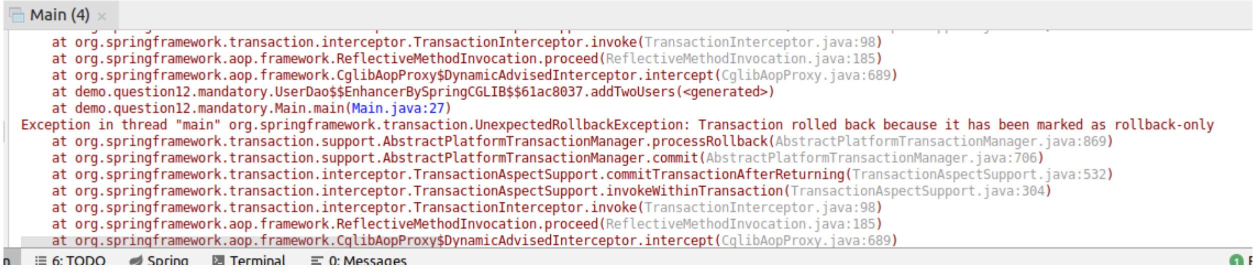
Output

When @Transactional is not present in UserDao.java method



```
Mar 14, 2019 4:44:36 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
Thu Mar 14 16:44:36 IST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.45+, 5.
org.springframework.transaction.IllegalTransactionStateException: No existing transaction found for transaction marked with propagation 'mandatory'
	at org.springframework.transaction.support.AbstractPlatformTransactionManager.getTransaction(AbstractPlatformTransactionManager.java:360)
	at org.springframework.transaction.interceptor.TransactionAspectSupport.createTransactionIfNecessary(TransactionAspectSupport.java:474)
	at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:289)
	at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:98)
```

When @Transactional is present, but exception is thrown in UserDao2.java, it rolls back both transactions, just like REQUIRED.



```
    at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:98)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:185)
    at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:689)
    at demo.question12.mandatory.UserDao$$EnhancerBySpringCGLIB$$61ac8037.addTwoUsers(<generated>)
    at demo.question12.mandatory.Main.main(Main.java:27)
Exception in thread "main" org.springframework.transaction.UnexpectedRollbackException: Transaction rolled back because it has been marked as rollback-only
    at org.springframework.transaction.support.AbstractPlatformTransactionManager.processRollback(AbstractPlatformTransactionManager.java:869)
    at org.springframework.transaction.support.AbstractPlatformTransactionManager.commit(AbstractPlatformTransactionManager.java:706)
    at org.springframework.transaction.interceptor.TransactionAspectSupport.commitTransactionAfterReturning(TransactionAspectSupport.java:532)
    at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:304)
    at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:98)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:185)
    at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:689)
```

NEVER

UserDao.java

```
package demo.question12.never;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdaonever")
public class UserDao {
  @Autowired
  JdbcTemplate jdbcTemplate;
  @Autowired
  UserDao2 userDao2;
  @Transactional
  public void addTwoUsers(Users user1,Users user2)
  {
      final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
      jdbcTemplate.update(query,new Object[]{user1.getUserName(), user1.getName(), user1.getPassword(),
user1.getAge(), user1.getDob()});
      try {
          userDao2.addUser(user2);
      }
      catch (Exception ex)
      {
          ex.printStackTrace();
      }


  }

}
```

UserDao2.java

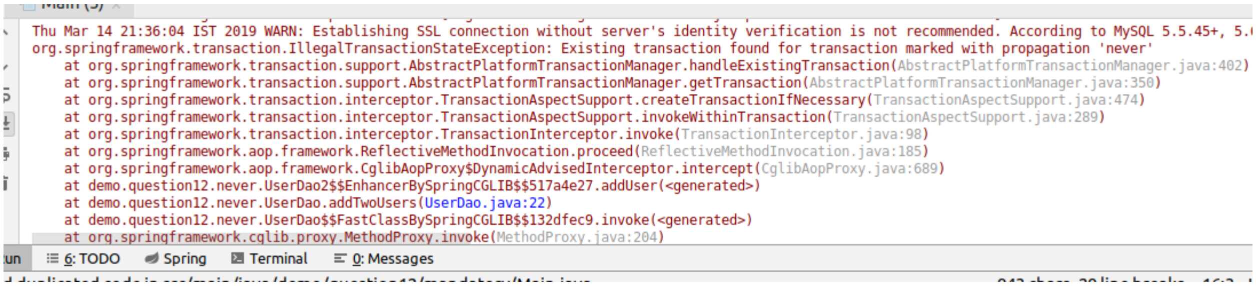
```
package demo.question12.never;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdao2never")
public class UserDao2 {
  @Autowired
  JdbcTemplate jdbcTemplate;
  @Transactional(propagation=Propagation.NEVER)
  public void addUser(Users user) {
      final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
      jdbcTemplate.update(query,new Object[]{user.getUserName(), user.getName(), user.getPassword(), 
user.getAge(), user.getDob()});
      throw new RuntimeException();
   }
}
```

Main.java

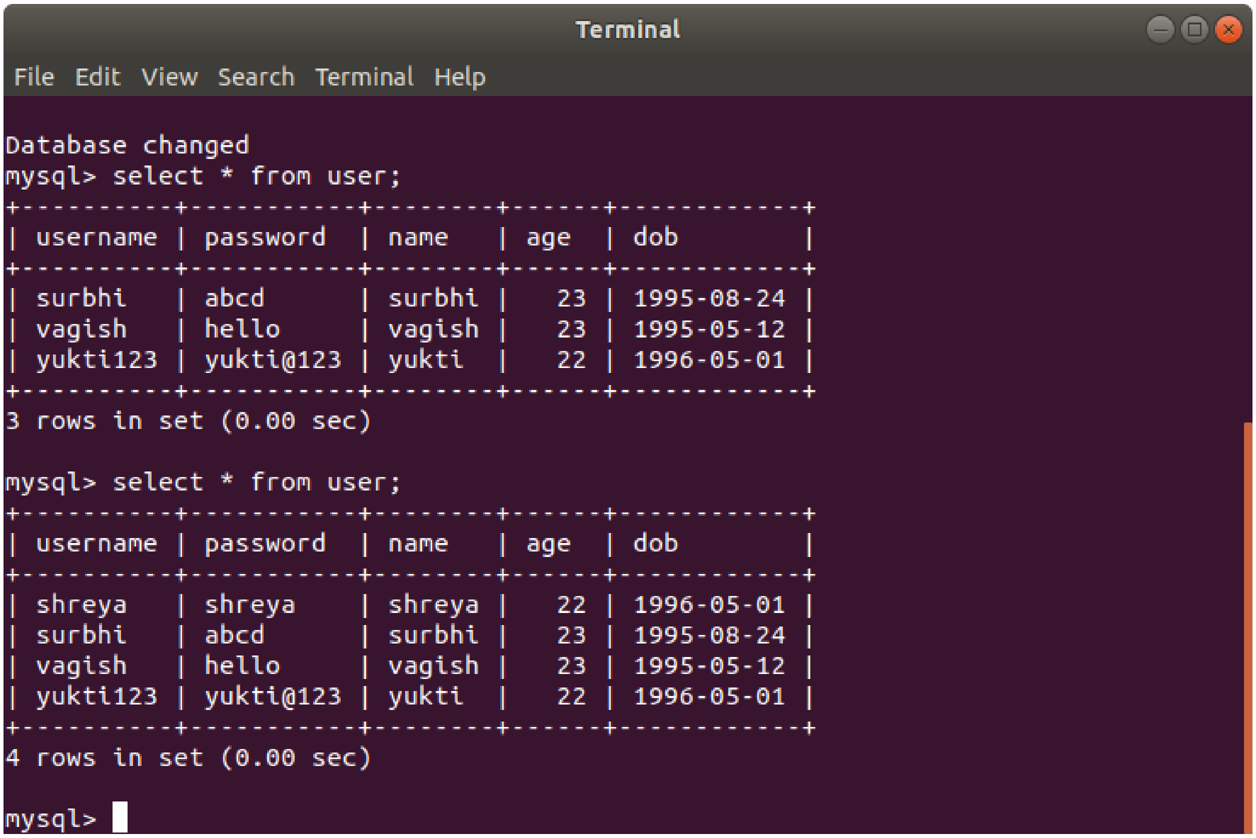
```
package demo.question12.never;

import demo.question5to11.Users;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.time.LocalDate;

public class Main {
   public static void main(String[] args) {
      ApplicationContext applicationContext=new ClassPathXmlApplicationContext("spring-config.xml");
     UserDao userDao= applicationContext.getBean(UserDao.class);
      Users user=new Users();
      user.setUserName("shreya");
      user.setName("shreya");
      user.setPassword("shreya");
      user.setAge(22);
      user.setDob(LocalDate.parse("1996-05-01"));

      Users user2=new Users();
      user2.setUserName("shikha");
      user2.setName("shikha");
      user2.setPassword("shikha");
      user2.setAge(23);
      user2.setDob(LocalDate.parse("1995-05-01"));

      userDao.addTwoUsers(user,user2);
   }
}
```

Inference

Doesn’t expect caller to be transactional method, so throws an exception if called inside method marked with @Transactional, but transactional method gets executed

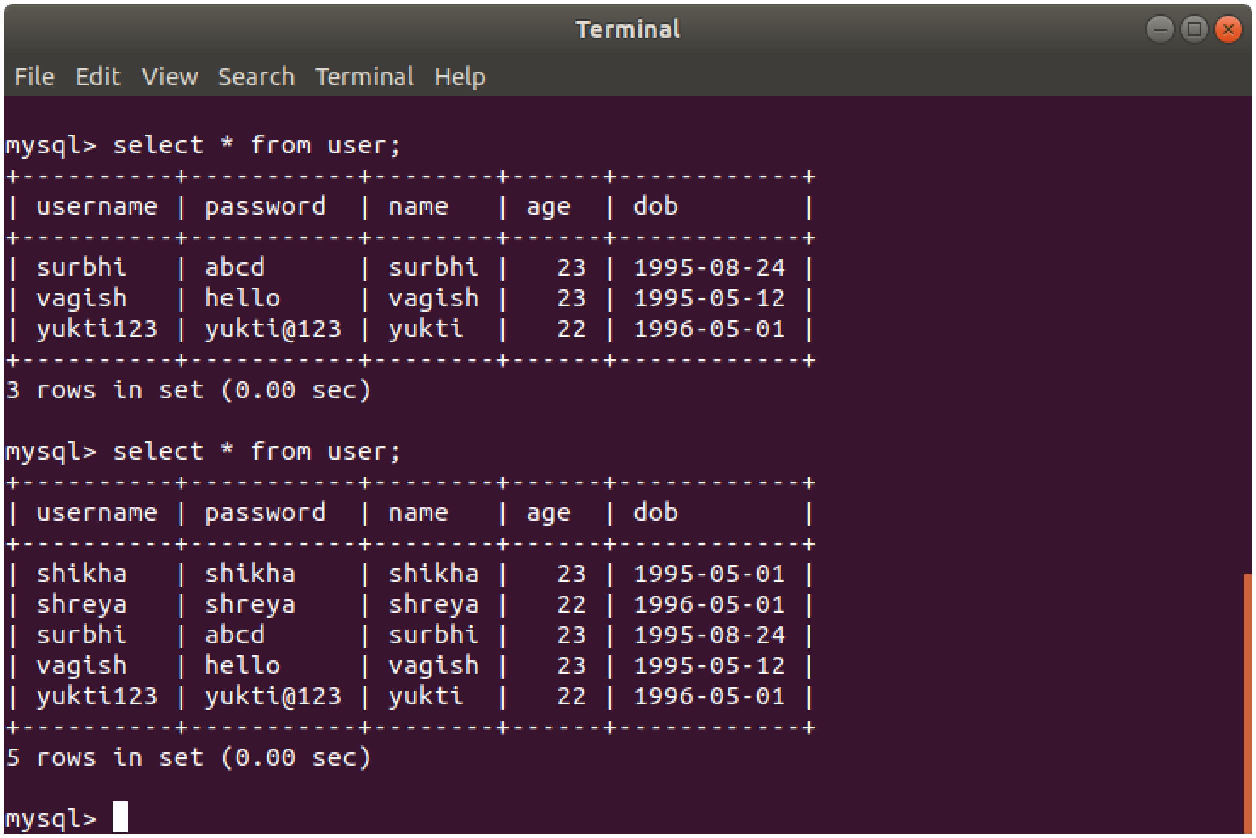


```
Thu Mar 14 21:36:04 IST 2019 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.45+, 5.(
org.springframework.transaction.IllegalTransactionStateException: Existing transaction found for transaction marked with propagation 'never'
    at org.springframework.transaction.support.AbstractPlatformTransactionManager.handleExistingTransaction(AbstractPlatformTransactionManager.java:402)
    at org.springframework.transaction.support.AbstractPlatformTransactionManager.getTransaction(AbstractPlatformTransactionManager.java:350)
    at org.springframework.transaction.interceptor.TransactionAspectSupport.createTransactionIfNecessary(TransactionAspectSupport.java:474)
    at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:289)
    at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:98)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:185)
    at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:689)
    at demo.question12.never.UserDao2$$EnhancerBySpringCGLIB$$517a4e27.addUser(<generated>)
    at demo.question12.never.UserDao.addTwoUsers(UserDao.java:22)
    at demo.question12.never.UserDao$$FastClassBySpringCGLIB$$132dfec9.invoke(<generated>)
    at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:204)
```



```
Database changed
mysql> select * from user;
+----------+-----------+--------+------+------------+
| username | password  | name   | age  | dob        |
+----------+-----------+--------+------+------------+
| surbhi   | abcd      | surbhi |   23 | 1995-08-24 |
| vagish   | hello     | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+-----------+--------+------+------------+
3 rows in set (0.00 sec)

mysql> select * from user;
+----------+-----------+--------+------+------------+
| username | password  | name   | age  | dob        |
+----------+-----------+--------+------+------------+
| shreya   | shreya    | shreya |   22 | 1996-05-01 |
| surbhi   | abcd      | surbhi |   23 | 1995-08-24 |
| vagish   | hello     | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+-----------+--------+------+------------+
4 rows in set (0.00 sec)

mysql> 
```

When @Transactional is removed from caller(in UserDao.java) and an exception is thrown in UserDao2.java, both users get saved,ie, works non-transactionally.



```
mysql> select * from user;
+----------+-----------+--------+------+------------+
| username | password  | name   | age  | dob        |
+----------+-----------+--------+------+------------+
| surbhi   | abcd      | surbhi |   23 | 1995-08-24 |
| vagish   | hello     | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+-----------+--------+------+------------+
3 rows in set (0.00 sec)

mysql> select * from user;
+----------+-----------+--------+------+------------+
| username | password  | name   | age  | dob        |
+----------+-----------+--------+------+------------+
| shikha   | shikha    | shikha |   23 | 1995-05-01 |
| shreya   | shreya    | shreya |   22 | 1996-05-01 |
| surbhi   | abcd      | surbhi |   23 | 1995-08-24 |
| vagish   | hello     | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+-----------+--------+------+------------+
5 rows in set (0.00 sec)

mysql> 
```

NOT_SUPPORTED

Execute non transactionally, suspend the current transaction if one exists

UserDao.java

```
package demo.question12.notsupported;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;


@Repository("userdaonotsupported")
public class UserDao {
  @Autowired
  JdbcTemplate jdbcTemplate;
  @Autowired
  UserDao2 userDao2;
 // @Transactional
  public void addTwoUsers(Users user1,Users user2)
  {
     final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
     jdbcTemplate.update(query,new Object[]{user1.getUserName(), user1.getName(), user1.getPassword(),
user1.getAge(), user1.getDob()});
     try {
        userDao2.addUser(user2);
     }
     catch (Exception ex)
     {
        ex.printStackTrace();
     }

          //       throw new RuntimeException();

  }

}
```

UserDao2.java

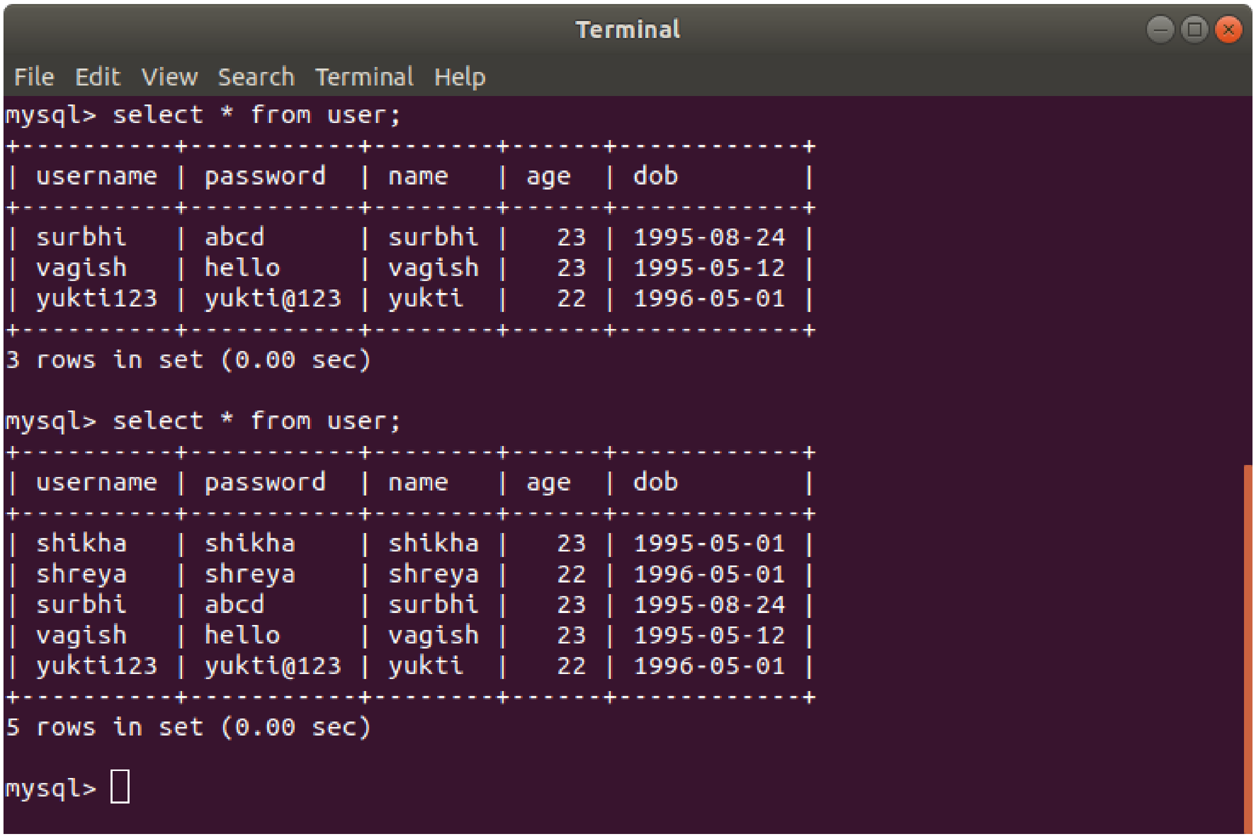
```
package demo.question12.notsupported;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdao2notsupported")
public class UserDao2 {
  @Autowired
  JdbcTemplate jdbcTemplate;
  @Transactional(propagation=Propagation.NOT_SUPPORTED)
  public void addUser(Users user) {
      final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
      jdbcTemplate.update(query,new Object[]{user.getUserName(), user.getName(), user.getPassword(),
user.getAge(), user.getDob()});
      throw new RuntimeException();
  }
}
```

Main.java

```
package demo.question12.notsupported;

import demo.question5to11.Users;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.time.LocalDate;

public class Main {
  public static void main(String[] args) {
      ApplicationContext applicationContext=new ClassPathXmlApplicationContext("spring-config.xml");
     UserDao userDao= applicationContext.getBean(UserDao.class);
      Users user=new Users();
      user.setUserName("shreya");
      user.setName("shreya");
      user.setPassword("shreya");
      user.setAge(22);
      user.setDob(LocalDate.parse("1996-05-01"));

      Users user2=new Users();
      user2.setUserName("shikha");
      user2.setName("shikha");
      user2.setPassword("shikha");
      user2.setAge(23);
      user2.setDob(LocalDate.parse("1995-05-01"));

      userDao.addTwoUsers(user,user2);
  }
}
```

Inference

Even if exception is thrown in UserDao2.java, both users get saved because it is executed in Non-Transactional form



```
mysql> select * from user;
+----------+----------+--------+------+------------+
| username | password | name   | age  | dob        |
+----------+----------+--------+------+------------+
| surbhi   | abcd     | surbhi |   23 | 1995-08-24 |
| vagish   | hello    | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti |   22 | 1996-05-01 |
+----------+----------+--------+------+------------+
3 rows in set (0.00 sec)

mysql> select * from user;
+----------+----------+--------+------+------------+
| username | password | name   | age  | dob        |
+----------+----------+--------+------+------------+
| shikha   | shikha   | shikha |   23 | 1995-05-01 |
| shreya   | shreya   | shreya |   22 | 1996-05-01 |
| surbhi   | abcd     | surbhi |   23 | 1995-08-24 |
| vagish   | hello    | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti |   22 | 1996-05-01 |
+----------+----------+--------+------+------------+
5 rows in set (0.00 sec)

mysql> 
```

Even if i uncomment throw new RuntimeException () in UserDao.java, that is not rolled back, since it gets suspended by NOT_SUPPORTED and both users get saved.

SUPPORTS

Supports current transaction, execute non-transactionally if none exists.

UserDao.java

```
package demo.question12.supports;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdaosupports")
public class UserDao {
   @Autowired
   JdbcTemplate jdbcTemplate;
   @Autowired
   UserDao2 userDao2;
   @Transactional
   public void addTwoUsers(Users user1,Users user2)
   {
       final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
       jdbcTemplate.update(query,new Object[]{user1.getUserName(), user1.getName(), user1.getPassword(),
user1.getAge(), user1.getDob()});
       try {
           userDao2.addUser(user2);
       }
       catch (Exception ex)
       {
           ex.printStackTrace();
       }


   }

}
```

UserDao2.java

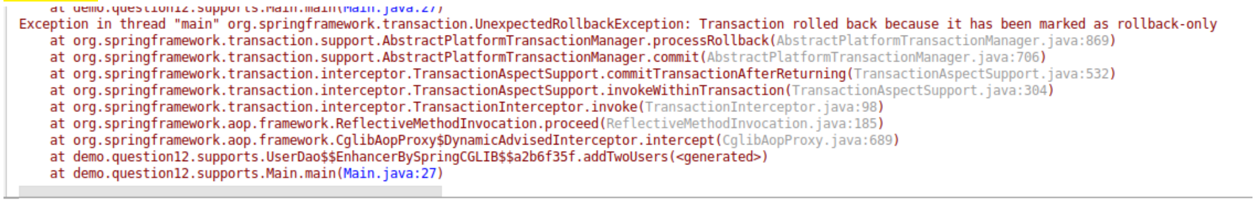
```
package demo.question12.supports;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdao2supports")
public class UserDao2 {
   @Autowired
   JdbcTemplate jdbcTemplate;
   @Transactional(propagation=Propagation.SUPPORTS)
   public void addUser(Users user) {
      final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
      jdbcTemplate.update(query,new Object[]{user.getUserName(), user.getName(), user.getPassword(),
user.getAge(), user.getDob()});
      throw new RuntimeException();
  }
}
```

Main.java

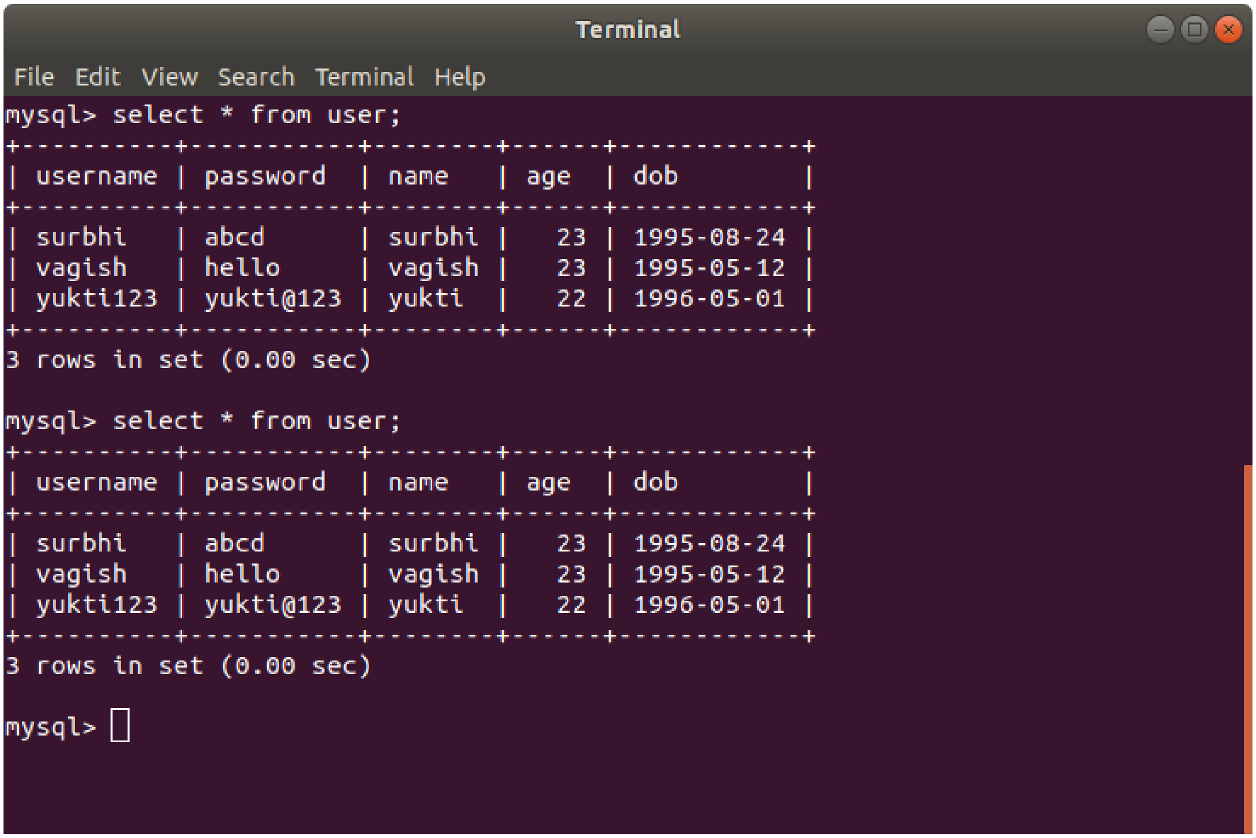
```
package demo.question12.supports;

import demo.question5to11.Users;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.time.LocalDate;

public class Main {
  public static void main(String[] args) {
      ApplicationContext applicationContext=new ClassPathXmlApplicationContext("spring-config.xml");
     UserDao userDao= applicationContext.getBean(UserDao.class);
      Users user=new Users();
      user.setUserName("shreya");
      user.setName("shreya");
      user.setPassword("shreya");
      user.setAge(22);
      user.setDob(LocalDate.parse("1996-05-01"));

      Users user2=new Users();
      user2.setUserName("shikha");
      user2.setName("shikha");
      user2.setPassword("shikha");
      user2.setAge(23);
      user2.setDob(LocalDate.parse("1995-05-01"));

      userDao.addTwoUsers(user,user2);
  }
}
```

Inference

When called within a transactional method, works like REQUIRE,i.e., no user gets saved.

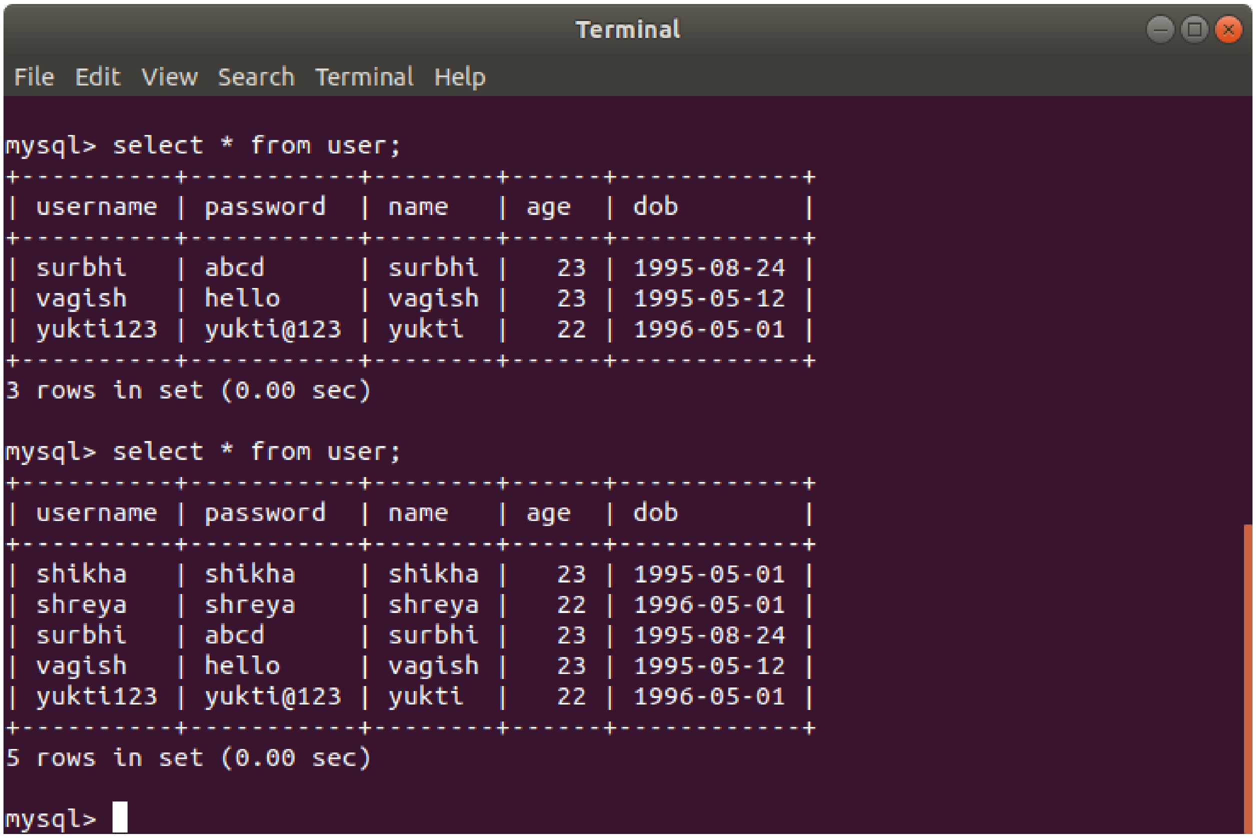


```
    at demo.question12.supports.Main.main(Main.java:27)
Exception in thread "main" org.springframework.transaction.UnexpectedRollbackException: Transaction rolled back because it has been marked as rollback-only
    at org.springframework.transaction.support.AbstractPlatformTransactionManager.processRollback(AbstractPlatformTransactionManager.java:869)
    at org.springframework.transaction.support.AbstractPlatformTransactionManager.commit(AbstractPlatformTransactionManager.java:706)
    at org.springframework.transaction.interceptor.TransactionAspectSupport.commitTransactionAfterReturning(TransactionAspectSupport.java:532)
    at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:304)
    at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:98)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:185)
    at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:689)
    at demo.question12.supports.UserDao$$EnhancerBySpringCGLIB$$a2b6f35f.addTwoUsers(<generated>)
    at demo.question12.supports.Main.main(Main.java:27)
```



```
mysql> select * from user;
+----------+-----------+--------+------+------------+
| username | password  | name   | age  | dob        |
+----------+-----------+--------+------+------------+
| surbhi   | abcd      | surbhi |   23 | 1995-08-24 |
| vagish   | hello     | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+-----------+--------+------+------------+
3 rows in set (0.00 sec)

mysql> select * from user;
+----------+-----------+--------+------+------------+
| username | password  | name   | age  | dob        |
+----------+-----------+--------+------+------------+
| surbhi   | abcd      | surbhi |   23 | 1995-08-24 |
| vagish   | hello     | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+-----------+--------+------+------------+
3 rows in set (0.00 sec)

mysql> 
```

When @Transactional is removed from UserDao.java, both users get saved.



```
mysql> select * from user;
+----------+-----------+--------+------+------------+
| username | password  | name   | age  | dob        |
+----------+-----------+--------+------+------------+
| surbhi   | abcd      | surbhi |   23 | 1995-08-24 |
| vagish   | hello     | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+-----------+--------+------+------------+
3 rows in set (0.00 sec)

mysql> select * from user;
+----------+-----------+--------+------+------------+
| username | password  | name   | age  | dob        |
+----------+-----------+--------+------+------------+
| shikha   | shikha    | shikha |   23 | 1995-05-01 |
| shreya   | shreya    | shreya |   22 | 1996-05-01 |
| surbhi   | abcd      | surbhi |   23 | 1995-08-24 |
| vagish   | hello     | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+-----------+--------+------+------------+
5 rows in set (0.00 sec)

mysql> 
```

Q13. Demonstrate the use of following options of @Transactional annotation

- read-only
- timeout
- rollback-for
- no-rollback-for

Solution

READ-ONLY

UserDao.java

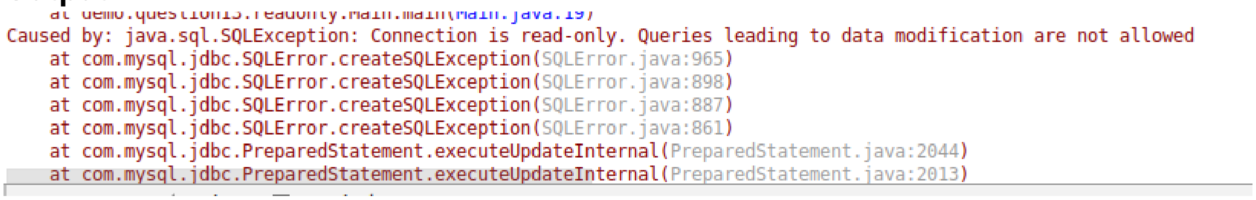
```
package demo.question13.readonly;

import demo.question12.mandatory.UserDao2;
import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdaoreadonly")
public class UserDao {
   @Autowired
   JdbcTemplate jdbcTemplate;
   @Transactional(readOnly = true)
   public void addUser(Users user1)
   {
       final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
       jdbcTemplate.update(query,new Object[]{user1.getUserName(), user1.getName(), user1.getPassword(),
user1.getAge(), user1.getDob()});
   }

}
```

Main.java

```
package demo.question13.readonly;

import demo.question5to11.Users;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.time.LocalDate;

public class Main {
   public static void main(String[] args) {
       ApplicationContext applicationContext=new ClassPathXmlApplicationContext("spring-config.xml");
       UserDao userDao=applicationContext.getBean(UserDao.class);
       Users user=new Users();
       user.setName("xyz");
       user.setUserName("xyz");
       user.setPassword("xyz");
       user.setAge(25);
       user.setDob(LocalDate.parse("1993-07-08"));
       userDao.addUser(user);
```

```
  }
}
```

Output



```
    at demo.question13.readonly.Main.main(Main.java:19)
Caused by: java.sql.SQLException: Connection is read-only. Queries leading to data modification are not allowed
    at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:965)
    at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:898)
    at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:887)
    at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:861)
    at com.mysql.jdbc.PreparedStatement.executeUpdateInternal(PreparedStatement.java:2044)
    at com.mysql.jdbc.PreparedStatement.executeUpdateInternal(PreparedStatement.java:2013)
```

TIMEOUT

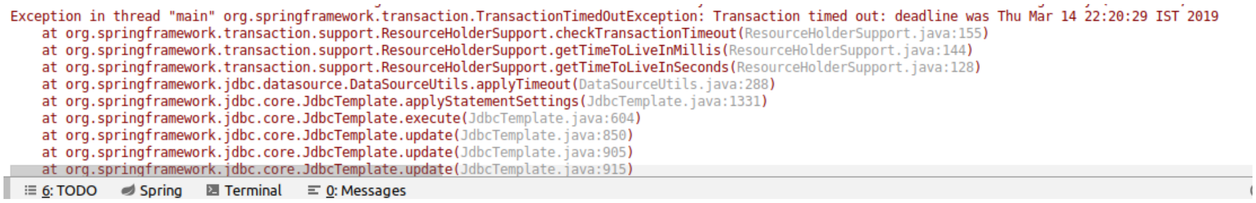
UserDao.java

```
package demo.question13.timeout;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdaotimeout")
public class UserDao {
  @Autowired
  JdbcTemplate jdbcTemplate;
  @Transactional(timeout = 2)
  public void addUser(Users user1)
  {
      try {
          Thread.sleep(3000L);
      } catch (InterruptedException e) {
          e.printStackTrace();
      }
      final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
      jdbcTemplate.update(query,new Object[]{user1.getUserName(), user1.getName(), user1.getPassword(),
user1.getAge(), user1.getDob()});
  }

}
```

Main.java

Same as demo.question13.rollback.Main

Output



```
Exception in thread "main" org.springframework.transaction.TransactionTimedOutException: Transaction timed out: deadline was Thu Mar 14 22:20:29 IST 2019
    at org.springframework.transaction.support.ResourceHolderSupport.checkTransactionTimeout(ResourceHolderSupport.java:155)
    at org.springframework.transaction.support.ResourceHolderSupport.getTimeToLiveInMillis(ResourceHolderSupport.java:144)
    at org.springframework.transaction.support.ResourceHolderSupport.getTimeToLiveInSeconds(ResourceHolderSupport.java:128)
    at org.springframework.jdbc.datasource.DataSourceUtils.applyTimeout(DataSourceUtils.java:288)
    at org.springframework.jdbc.core.JdbcTemplate.applyStatementSettings(JdbcTemplate.java:1331)
    at org.springframework.jdbc.core.JdbcTemplate.execute(JdbcTemplate.java:604)
    at org.springframework.jdbc.core.JdbcTemplate.update(JdbcTemplate.java:850)
    at org.springframework.jdbc.core.JdbcTemplate.update(JdbcTemplate.java:905)
    at org.springframework.jdbc.core.JdbcTemplate.update(JdbcTemplate.java:915)
```

ROLLBACKFOR

UserDao.java

```
package demo.question13.rollbackfor;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdaorollbackfor")
public class UserDao {
  @Autowired
  JdbcTemplate jdbcTemplate;
  @Transactional(rollbackFor = RuntimeException.class)
  public void addUser(Users user1)
  {
      final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
      jdbcTemplate.update(query,new Object[]{user1.getUserName(), user1.getName(), user1.getPassword(),
user1.getAge(), user1.getDob()});
      throw new RuntimeException();
  }

}
```

Main.java

Same as demo.question13.rollback.Main

Inference

Record with username xyz will not be saved

NoRollBackFor

UserDao.java

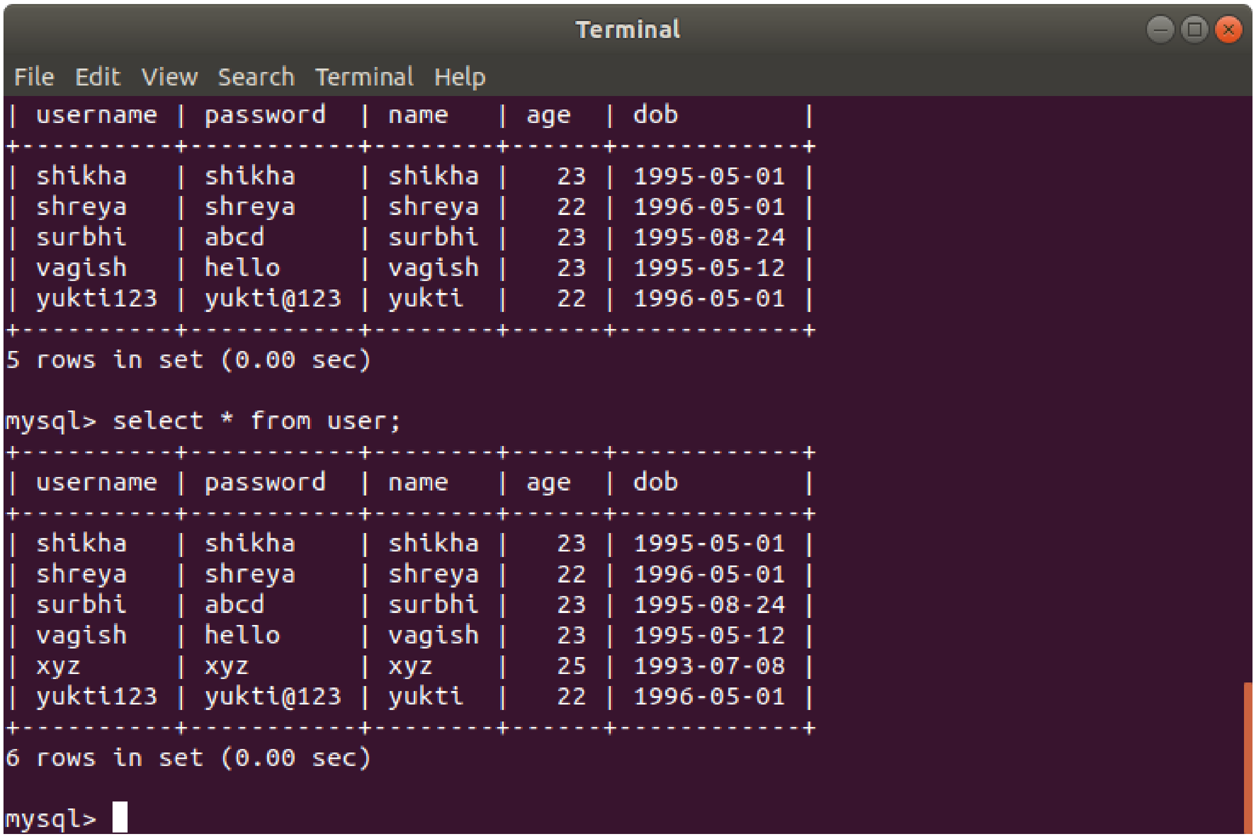
```
package demo.question13.norollbackfor;

import demo.question5to11.Users;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;


@Repository("userdaonorollbackfor")
public class UserDao {
  @Autowired
  JdbcTemplate jdbcTemplate;
  @Transactional(noRollbackFor = RuntimeException.class)
  public void addUser(Users user1)
  {
     final  String query="insert into user(username,name,password,age,dob)values(?,?,?,?,?)";
     jdbcTemplate.update(query,new Object[]{user1.getUserName(), user1.getName(), user1.getPassword(), user1.getAge(), user1.getDob()});
     throw new RuntimeException();
  }

}
```

Main.java

Same as demo.question13.rollback.Main

Inference

User with username xyz gets saved, even after exception occurs.



```
| username | password  | name   | age  | dob        |
+----------+-----------+--------+------+------------+
| shikha   | shikha    | shikha |   23 | 1995-05-01 |
| shreya   | shreya    | shreya |   22 | 1996-05-01 |
| surbhi   | abcd      | surbhi |   23 | 1995-08-24 |
| vagish   | hello     | vagish |   23 | 1995-05-12 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+-----------+--------+------+------------+
5 rows in set (0.00 sec)

mysql> select * from user;
+----------+-----------+--------+------+------------+
| username | password  | name   | age  | dob        |
+----------+-----------+--------+------+------------+
| shikha   | shikha    | shikha |   23 | 1995-05-01 |
| shreya   | shreya    | shreya |   22 | 1996-05-01 |
| surbhi   | abcd      | surbhi |   23 | 1995-08-24 |
| vagish   | hello     | vagish |   23 | 1995-05-12 |
| xyz      | xyz       | xyz    |   25 | 1993-07-08 |
| yukti123 | yukti@123 | yukti  |   22 | 1996-05-01 |
+----------+-----------+--------+------+------------+
6 rows in set (0.00 sec)

mysql> 
```