Creación de base de datos en Azure SQL Database y su conexión a un proyecto Java

Requisitos Previos:

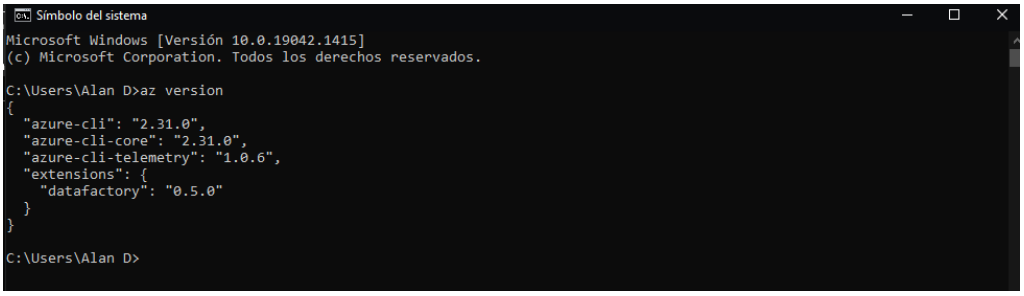
- Spring Tool Suite 4
- Azure CLI
- Cuenta en Azure con una suscripción activa.
- Java Development Kit versión 8 u 11
- Apache Maven 3.0 o posterior

Instrucciones:

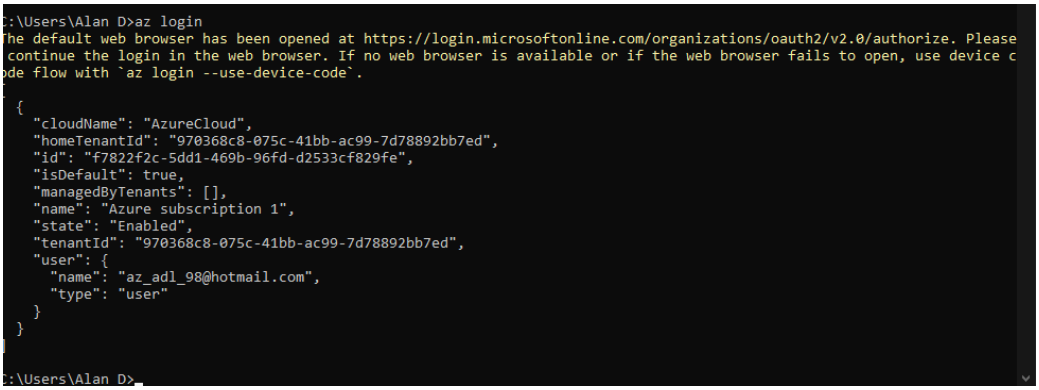
Azure SQL Database es un servicio totalmente administrado de bases de datos en la nube de Microsoft. Se encarga de la mayoría de las funciones de administración de bases de datos, como actualizar, aplicar revisiones, crear copias de seguridad y supervisar sin intervención del usuario. SQL Database puede ser la opción adecuada para una variedad de aplicaciones modernas en la nube, porque le permite procesar tanto datos relacionales como estructuras no relacionales.

Empecemos utilizando la interfaz de la línea de comandos de Azure (CLI de Azure), una guía para su instalación puede ser consultada en este vínculo.

Una vez instalado, en una terminal de comandos de nuestro sistema operativo (Símbolos del Sistema de Windows para este ejemplo), verificaremos la correcta instalación de la herramienta Azure CLI escribiendo el comando **AZ VERSION.** Obtendremos un resultado como el siguiente:



Iniciaremos sesión en nuestra cuenta de Azure con el comando **AZ LOGIN.** Una ventana del navegador determinado se abrirá presentándonos instrucciones para iniciar sesión. Una vez terminado el proceso de iniciar sesión, en la terminal de comandos aparecerá lo siguiente:

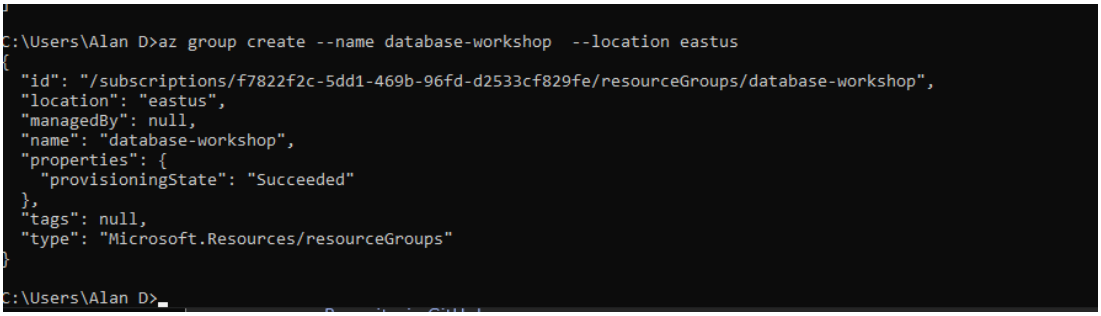


Ya estamos conectados a Azure, ahora el primer paso para la creación de una base de daros SQL en Azure es crear un “grupo de recursos”, que será el espacio dentro de nuestra suscripción en donde estarán alojados todos los recursos que utilizaremos para este ejemplo. Para su creación necesitaremos escribir el siguiente comando:

az group create --name database-workshop --location eastus

Se puede sustituir “**database-workshop**” por cualquier nombre.

La localización por default es “eastus” pero puede ser cambiado según sea la necesidad, ver lista de ubicaciones con el comando **az account list-locations -o table**



Crearemos nuestro servidor de SQL dentro de nuestro grupo de recursos con el siguiente comando:

az sql server create --resource-group database-workshop --name azpadsasql --location eastus --admin-user demo --admin-password yourpass

El nombre del servidor tiene que ser único e irrepetible, tener solo letras minúsculas, número o “-” y no puede tener una longitud mayor a 63 caracteres.

Es importante tener presentes el nombre de usuario declarado en “admin-user” y la contraseña de “admin-password” para su uso posterior.

Para evitar problemas posteriores de conexión, crearemos una regla para que el firewall de Azure permita la conexión desde nuestra dirección ip. Requeriremos conocer la dirección ip designada a

nuestro equipo de trabajo, para conocerlo podemos consultar la página: http://whatismyip.akamai.com

Colocamos el siguiente comando:

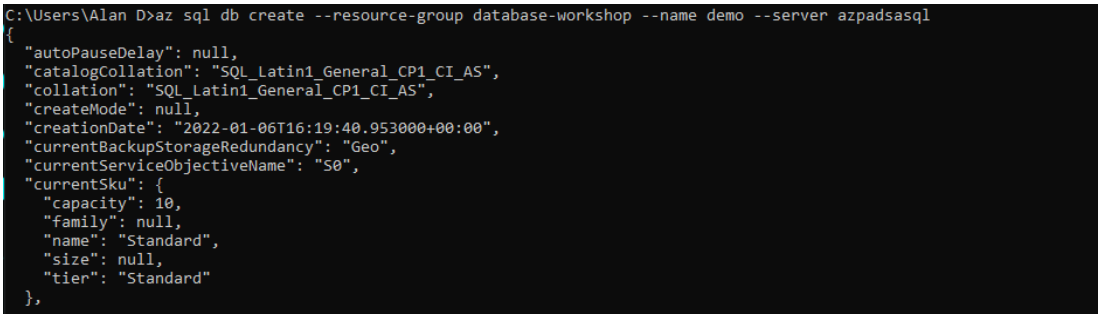
az sql server firewall-rule create --resource-group database-workshop --name azpadsasql-database-allow-local-ip --server azpadsasql --start-ip-address 192.0.0.0 --end-ip-address 192.0.0.0

--name: puede ser un nombre cualquiera

La ip de inicio y de final pueden ser las mismas.

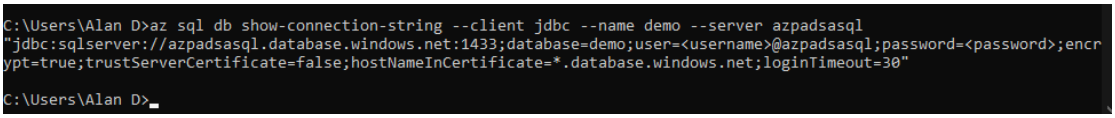
Ahora, podemos crear nuestra base de datos con el siguiente comando:

az sql db create --resource-group database-workshop --name demo --server azpadsasql



Requeriremos una cadena de conexión de nuestra base de datos para utilizarlo en el proyecto Java, para obtenerlo utilizaremos el siguiente comando:

az sql db show-connection-string --client jdbc --name demo --server azpadsasql

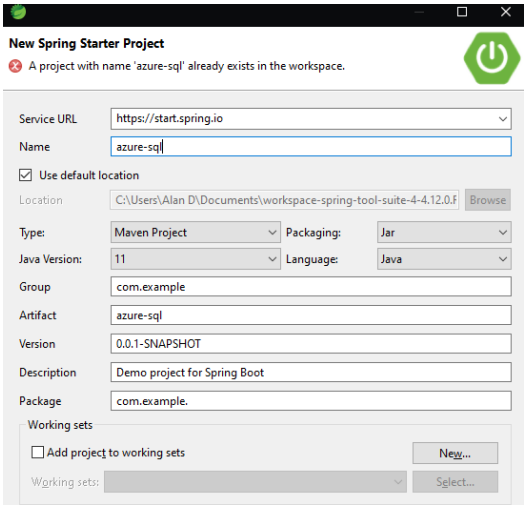


El cliente que utilizaremos para Java será jdbc (Java Database Connectivity)

Hemos creado con éxito nuestro servidor y base de datos SQL en una suscripción de Azure.

A continuación, realizaremos un proyecto utilizando Spring Boot Suite 4 para facilitar la creación de proyectos Java que requieran dependencias y propiedades de la aplicación.

Creamos un nuevo proyecto de Spring (New Spring Starter Project) con las siguientes configuraciones:



Gracias a Sprint, tenemos generados automáticamente los archivos pom.xml y el recurso application.properties, los cuales serán los lugares donde declararemos nuestras dependencias de Maven y los parámetros de conexión a nuestra base de datos en Azure.

En el archivo pom.xml, agregaremos las siguientes dependencias:

```
    <groupId>com.example</groupId>
    <artifactId>azure-sql</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>azure-sql</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>11</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.microsoft.sqlserver</groupId>
            <artifactId>mssql-jdbc</artifactId>
            <version>7.4.1.jre11</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

En el recurso application.properties (src/main/resources) pondremos la cadena de conexión que obtuvimos anteriormente:

```
url=jdbc:sqlserver://azpadsasql.database.windows.net:1433;database=demo;user=<username>@azpadsasql;password=<password>;encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.database.windows.net;loginTimeout=30;
```

Cambiar `<username> y <password>;` por el usuario y contraseña declarados durante la creación del servidor SQL.

Después, crearemos dentro de la carpeta src/main/resources un nuevo archivo (file) llamado schema.sql donde pondremos el siguiente contenido:

```
DROP TABLE IF EXISTS todo;
CREATE TABLE todo (id INT PRIMARY KEY, description VARCHAR(255), details VARCHAR(4096), done BIT);
```

Creamos una nueva clase llamada Todo.java en la carpeta src/main/java, puede estar dentro de la carpeta generada por Sprint (com.example). En esta clase funcionará como nuestro modelo donde declararemos las variables descritas en el esquema SQL (int, description, details, done) con sus respectivos get y set así como el constructor.

```
package com.example;

        public class Todo {

            private Long id;
            private String description;
            private String details;
            private boolean done;

            public Todo() {
            }

            public Todo(Long id, String description, String details, boolean done) {
                this.id = id;
                this.description = description;
                this.details = details;
                this.done = done;
            }

            public Long getId() {
                return id;
            }

            public void setId(Long id) {
                this.id = id;
            }

            public String getDescription() {
                return description;
            }

            public void setDescription(String description) {
                this.description = description;
            }

            public String getDetails() {
```

```
        return details;
    }

    public void setDetails(String details) {
        this.details = details;
    }

    public boolean isDone() {
        return done;
    }

    public void setDone(boolean done) {
        this.done = done;
    }

    @Override
    public String toString() {
        return "Todo{" +
                "id=" + id +
                ", description='" + description + '\'' +
                ", details='" + details + '\'' +
                ", done=" + done +
                '}';
    }
}
```

Posteriormente, en la clase principal del proyecto (AzureSqlApplication.java) escribiremos el siguiente código el cual nos dará información sobre si la conexión a la base de datos fue exitosa.

```
package com.example;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import java.sql.*;
import java.util.*;
import java.util.logging.Logger;

@SpringBootApplication
public class AzureSqlApplication {

    private static final Logger log;

    static {
        System.setProperty("java.util.logging.SimpleFormatter.format", "[%4$-7s] %5$s %n");
        log =Logger.getLogger(AzureSqlApplication.class.getName());
    }

    public static void main(String[] args) throws Exception {
        log.info("Loading application properties");
        Properties properties = new Properties();

properties.load(AzureSqlApplication.class.getClassLoader().getResourceAsStream("application.properti
es"));

        log.info("Connecting to the database");
        Connection connection = DriverManager.getConnection(properties.getProperty("url"),
properties);
        log.info("Database connection test: " + connection.getCatalog());

        log.info("Create database schema");
        Scanner scanner = new
Scanner(AzureSqlApplication.class.getClassLoader().getResourceAsStream("schema.sql"));
        Statement statement = connection.createStatement();
        while (scanner.hasNextLine()) {
            statement.execute(scanner.nextLine());
```

```
            }

            log.info("Closing database connection");
            connection.close();
    }

}
```

```
@SpringBootApplication
public class AzureSqlApplication {

        private static final Logger log;

        static {
            System.setProperty("java.util.logging.SimpleFormatter.format", "[%4$-7s] %5$s %n");
            log =Logger.getLogger(AzureSqlApplication.class.getName());
        }

        public static void main(String[] args) throws Exception {
            log.info("Loading application properties");
            Properties properties = new Properties();
            properties.load(AzureSqlApplication.class.getClassLoader().getResourceAsStream("application.properties"));

            log.info("Connecting to the database");
            Connection connection = DriverManager.getConnection(properties.getProperty("url"), properties);
            log.info("Database connection test: " + connection.getCatalog());

            log.info("Create database schema");
            Scanner scanner = new Scanner(AzureSqlApplication.class.getClassLoader().getResourceAsStream("schema.sql"));
            Statement statement = connection.createStatement();
            while (scanner.hasNextLine()) {
                statement.execute(scanner.nextLine());
            }

Problems  Javadoc  Declaration  Search  Console  Progress
<terminated> AzureSqlApplication [Java Application] C:\sts4\sts-4.12.0.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v20210721-1149\jre\bi
[INFO   ] Loading application properties
[INFO   ] Connecting to the database
[INFO   ] Database connection test: demo
[INFO   ] Create database schema
[INFO   ] Closing database connection
```

En este punto, ya estamos conectados con nuestra base de datos de Azure.

Haremos un pequeño ejemplo de inserción, consulta, actualización y borrado de datos.

Primero escribiremos el siguiente código después del bucle While y antes de `log.info("Closing database connection");`

```
        Todo todo = new Todo(1L, "configuration", "congratulations, you have set up JDBC
correctly!", true);
            insertData(todo, connection);

            todo = readData(connection);

            todo.setDetails("congratulations, you have updated data!");
            updateData(todo, connection);

            deleteData(todo, connection);
```

Comentaremos el código agregado obteniendo un resultado como el siguiente:

```
    private static final Logger log;

    static {
        System.setProperty("java.util.logging.SimpleFormatter.format", "[%4$-7s] %5$s %n");
        log =Logger.getLogger(AzureSqlApplication.class.getName());
    }

    public static void main(String[] args) throws Exception {
        log.info("Loading application properties");
        Properties properties = new Properties();
        properties.load(AzureSqlApplication.class.getClassLoader().getResourceAsStream("application.properties"));

        log.info("Connecting to the database");
        Connection connection = DriverManager.getConnection(properties.getProperty("url"), properties);
        log.info("Database connection test: " + connection.getCatalog());

        log.info("Create database schema");
        Scanner scanner = new Scanner(AzureSqlApplication.class.getClassLoader().getResourceAsStream("schema.sql"));
        Statement statement = connection.createStatement();
        while (scanner.hasNextLine()) {
            statement.execute(scanner.nextLine());
        }

//      Todo todo = new Todo(1L, "configuration", "congratulations, you have set up JDBC correctly!", true);
//      insertData(todo, connection);
//
//      todo = readData(connection);
//
//      todo.setDetails("congratulations, you have updated data!");
//      updateData(todo, connection);
//
//      deleteData(todo, connection);


        log.info("Closing database connection");
        connection.close();
    }
```

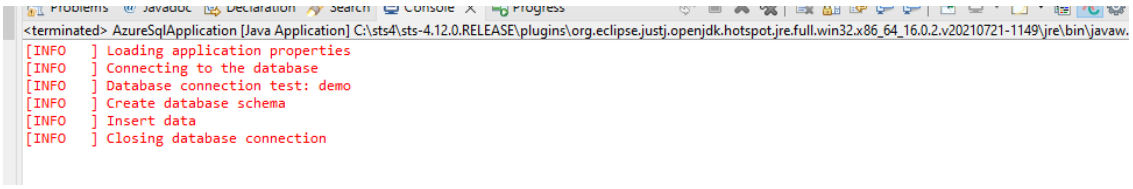
Entre las dos ultimas llaves de nuestro código, agregaremos un método que será el encargado de agregar datos a nuestra base de datos.

```
private static void insertData(Todo todo, Connection connection) throws SQLException {
                log.info("Insert data");
                PreparedStatement insertStatement = connection
                        .prepareStatement("INSERT INTO todo (id, description, details, done) VALUES
(?, ?, ?, ?);");

                insertStatement.setLong(1, todo.getId());
                insertStatement.setString(2, todo.getDescription());
                insertStatement.setString(3, todo.getDetails());
                insertStatement.setBoolean(4, todo.isDone());
                insertStatement.executeUpdate();
            }
```

Quitaremos los comentarios de las dos primeras líneas de código comentadas obteniendo un esquema como el siguiente:

```
            log.info("Connecting to the database");
            Connection connection = DriverManager.getConnection(properties.getProperty("url"), properties);
            log.info("Database connection test: " + connection.getCatalog());

            log.info("Create database schema");
            Scanner scanner = new Scanner(AzureSqlApplication.class.getClassLoader().getResourceAsStream("schema.sql"));
            Statement statement = connection.createStatement();
            while (scanner.hasNextLine()) {
                statement.execute(scanner.nextLine());
            }

            Todo todo = new Todo(1L, "configuration", "congratulations, you have set up JDBC correctly!", true);
            insertData(todo, connection);

//          todo = readData(connection);
//
//          todo.setDetails("congratulations, you have updated data!");
//          updateData(todo, connection);
//
//          deleteData(todo, connection);


            log.info("Closing database connection");
            connection.close();
        }

        private static void insertData(Todo todo, Connection connection) throws SQLException {
            log.info("Insert data");
            PreparedStatement insertStatement = connection
                    .prepareStatement("INSERT INTO todo (id, description, details, done) VALUES (?, ?, ?, ?);");

            insertStatement.setLong(1, todo.getId());
            insertStatement.setString(2, todo.getDescription());
            insertStatement.setString(3, todo.getDetails());
            insertStatement.setBoolean(4, todo.isDone());
            insertStatement.executeUpdate();
        }
    }
```

Podemos probar el código y obtendremos el siguiente resultado:



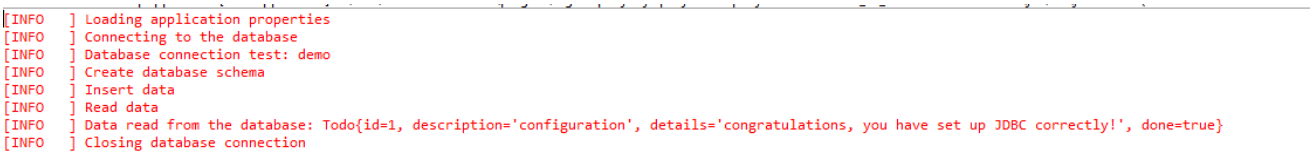
```
<terminated> AzureSqlApplication [Java Application] C:\sts4\sts-4.12.0.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v20210721-1149\jre\bin\javaw.
[INFO   ] Loading application properties
[INFO   ] Connecting to the database
[INFO   ] Database connection test: demo
[INFO   ] Create database schema
[INFO   ] Insert data
[INFO   ] Closing database connection
```

Ahora agregaremos código que será el encargado de leer los datos almacenados en la base de datos. Este código irá después del método realizado anteriormente:

```
        private static Todo readData(Connection connection) throws SQLException {
            log.info("Read data");
            PreparedStatement readStatement = connection.prepareStatement("SELECT * FROM
todo;");
            ResultSet resultSet = readStatement.executeQuery();
            if (!resultSet.next()) {
                log.info("There is no data in the database!");
                return null;
            }
            Todo todo = new Todo();
            todo.setId(resultSet.getLong("id"));
            todo.setDescription(resultSet.getString("description"));
            todo.setDetails(resultSet.getString("details"));
            todo.setDone(resultSet.getBoolean("done"));
            log.info("Data read from the database: " + todo.toString());
            return todo;
        }
```

Quitamos los comentarios de: todo = readData(connection);

Probamos el código y obtenemos el siguiente resultado:



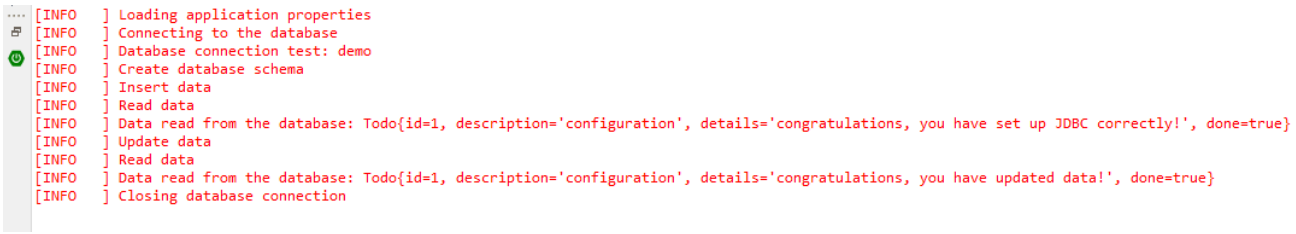
Ahora, colocaremos después del método realizado anteriormente, el siguiente código que nos permitirá actualizar los datos ingresados:

```
    private static void updateData(Todo todo, Connection connection) throws SQLException {
            log.info("Update data");
            PreparedStatement updateStatement = connection
                  .prepareStatement("UPDATE todo SET description = ?, details = ?, done = ?
WHERE id = ?;");

            updateStatement.setString(1, todo.getDescription());
            updateStatement.setString(2, todo.getDetails());
            updateStatement.setBoolean(3, todo.isDone());
            updateStatement.setLong(4, todo.getId());
            updateStatement.executeUpdate();
            readData(connection);
        }
```

Quitamos los comentarios de las líneas:

```
            todo.setDetails("congratulations, you have updated data!");
            updateData(todo, connection);
```

Obtendremos el siguiente resultado:



Para realizar un borrado de los datos, colocaremos el siguiente código después del método realizado anteriormente:

```
private static void deleteData(Todo todo, Connection connection) throws SQLException {
            log.info("Delete data");
            PreparedStatement deleteStatement = connection.prepareStatement("DELETE
FROM todo WHERE id = ?;");
            deleteStatement.setLong(1, todo.getId());
            deleteStatement.executeUpdate();
            readData(connection);
        }
```

Quitamos los comentarios de las líneas:

```
//              deleteData(todo, connection);
```

Obtendremos el siguiente resultado, el programa borrará los datos almacenados, realizará una nueva lectura de datos e informará que no hay datos en la base de datos:

```
[INFO   ] Loading application properties
[INFO   ] Connecting to the database
[INFO   ] Database connection test: demo
[INFO   ] Create database schema
[INFO   ] Insert data
[INFO   ] Read data
[INFO   ] Data read from the database: Todo{id=1, description='configuration', details='congratulations, you have set up JDBC correctly!', done=true}
[INFO   ] Update data
[INFO   ] Read data
[INFO   ] Data read from the database: Todo{id=1, description='configuration', details='congratulations, you have updated data!', done=true}
[INFO   ] Delete data
[INFO   ] Read data
[INFO   ] There is no data in the database!
[INFO   ] Closing database connection
```

Repositorio GitHub:

El proyecto generado en este ejemplo puede ser descargado en el siguiente repositorio de GitHub:

https://github.com/PADSA-github/AzureSQLDatabaseJava