PROGRAMACIÓN N-CAPAS LABORATORIO #8

JdbcTemplate

Catedrático:
Lic. Juan Lozano

Instructores:

Karla Beatriz Morales Alfaro 00022516@uca.edu.sv
Sara Noemy Romero Menjivar 00030716@uca.edu.sv
Salvador Edgardo Campos Gómez 00117716@uca.edu.sv

JdbcTemplate

Indicaciones generales:

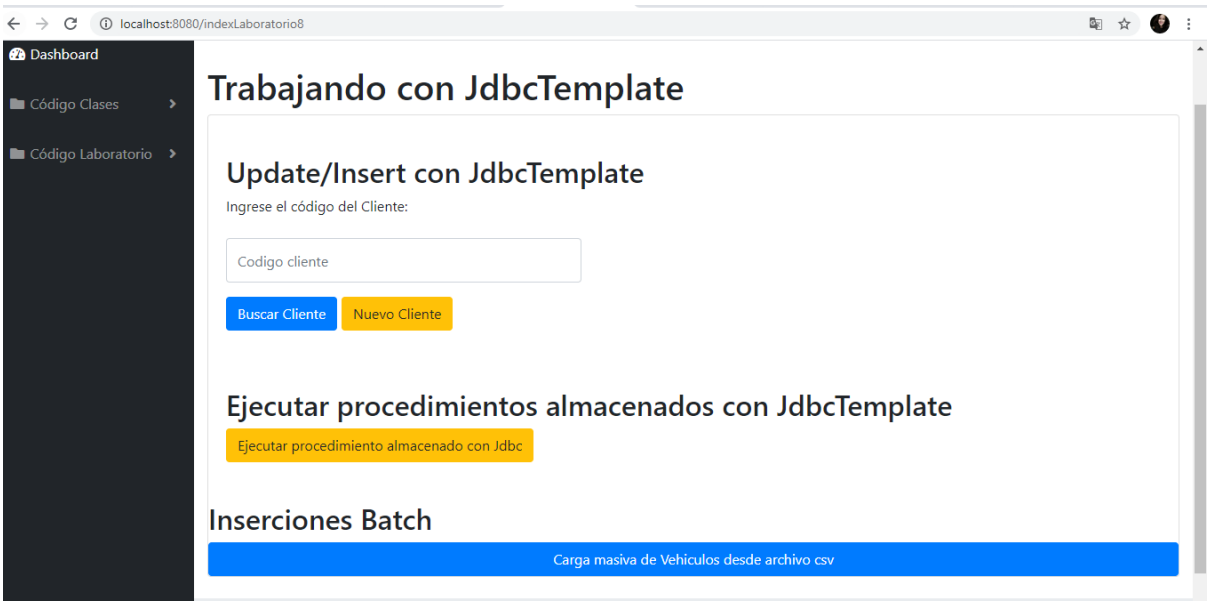
- Antes de comenzar el laboratorio se necesita clonar el siguiente repositorio para poder trabajar sobre el **(Trabajado sobre el código de clases)**:

  https://github.com/KarlaMorales97/Laboratorio-8-PNC

- Hacer "Restore" de **ventasng.backup** que se encuentra en **"Recursos de clase > Recursos > ventasng.backup".**
- Percatarse que el archivo **vehiculos.csv** se ha descargado correctamente.
- Todos los .html ya están configurados, solo se deberá seguir la guía y configurar las capas: **DAO, service y controller,** por lo tanto, debe asegurarse de nombrar los métodos de manera correcta.

Antes de empezar

Luego de haber clonado el repositorio, correr la aplicación e ingresar a http://localhost:8080/indexLaboratorio8 y asegurarse que se vea de la siguiente manera:



JdbcTemplate - Insert

----------CAPA DAO----------

La clase **SimpleJdbcInsert** nos provee de una forma fácil de realizar inserciones través de Jdbc (con JdbcTemplate).

En el constructor recibirá el objeto jdbcTemplate referenciado, luego con el método **.withSchemaName** y **.withTableName** recibirá el nombre del esquema/tabla respectivamente.

Esta clase permitirá hacer inserts a nivel de JDBC

Ejercicio práctico

Figura 1. ClienteDAO.

```
public int insertClienteAutoId(Cliente c);
```

Figura 1.1. ClienteDAOImpl.

```
  //JDBC
  @Autowired
  JdbcTemplate jdbcTemplate;

@Override
public int insertClienteAutoId(Cliente c) {
    // TODO Auto-generated method stub
            SimpleJdbcInsert jdbcInsert = new SimpleJdbcInsert(jdbcTemplate)
                    .withSchemaName("store")
                    .withTableName("cliente")
                    //PK
                    .usingGeneratedKeyColumns("c_cliente");

            //Valores del insert
            Map<String, Object> parametros = new HashMap<String, Object>();
            parametros.put("s_nombres", c.getSnombres());
            parametros.put("s_apellidos", c.getSapellidos());
            parametros.put("f_nacimiento", c.getFnacimiento());
            parametros.put("b_activo", c.getBactivo());

            //El metodo executeAndReturnKey devuelve la llave primaria generada en el insert
            Number id_generated = jdbcInsert.executeAndReturnKey(parametros);

            return id_generated.intValue();
}
```

Si el id ya existe en la base de datos, se procede a actualizar el registro.

Updates con JdbcTemplate

- Se hace la sentencia SQL del update.
- Se utiliza el método update, que recibe:
    - Sentencia SQL.
    - Valores de los parámetros.

Figura 1.2. ClienteDAO.

```
public void updateCliente(Cliente c);
```

Figura 1.3. ClienteDAOImpl.

```
  private static final String sql = "UPDATE store.cliente SET s_nombres = ?, s_apellidos = ?, f_nacimiento = ?, b_activo = ? WHERE c_cliente = ?";


  @Override
  public void updateCliente(Cliente c) {
      Object[] parametros = new Object[] {c.getSnombres(), c.getSapellidos(), c.getFnacimiento(), c.getBactivo(), c.getCcliente()};
      jdbcTemplate.update(sql, parametros);

  }
```

----------CAPA DE SERVICIO----------

Figura 2. ClienteService.

```
public int insertClienteAutoId(Cliente c);

public void updateCliente(Cliente c);
```

Figura 2.1 ClienteServiceImpl.

```
@Override
public int insertClienteAutoId(Cliente c) {
    // TODO Auto-generated method stub
    return clienteDao.insertClienteAutoId(c);
}

@Override
public void updateCliente(Cliente c) {
    // TODO Auto-generated method stub
    clienteDao.updateCliente(c);
}
```

----------CAPA CONTROLADOR----------

Figura 3 Laboratorio8Controller.

```
@Controller
public class Laboratorio8Controller {

    @Autowired
    private ClienteService clienteService;

    private static final Logger logger = Logger.getLogger(Laboratorio8Controller.class.getName());


    //MENU PRINCIPAL LABORATORIO 8
    @RequestMapping("/indexLaboratorio8")
    public ModelAndView indexLaboratorio8() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("Laboratorio/indexLaboratorio8");
        return mav;
    }

    //iNSERTAR CLIENTE
    @RequestMapping("/insertcliente")
    public ModelAndView nuevoCliente() {
        ModelAndView mav = new ModelAndView();
        mav.addObject("cliente", new Cliente());
        mav.setViewName("Laboratorio/agregarCliente");
        return mav;
    }
```

Figura 3.1 Laboratorio8Controller.

```
@RequestMapping(value="/savecliente", method = RequestMethod.POST)
public ModelAndView saveCliente(@Valid @ModelAttribute("cliente") Cliente c, BindingResult r){
    ModelAndView mav = new ModelAndView();
    mav.addObject("cliente", new Cliente());
    if(r.hasErrors()){
        mav.addObject("resultado", 0);
        mav.setViewName("Laboratorio/agregarCliente");
    }
    else{
        Integer key = null;
        if(c.getCcliente() == null) { //Insert
            mav.addObject("resultado", 1);
            key = clienteService.insertClienteAutoId(c);
        }
        else { //Update
            mav.addObject("resultado", 1);
            clienteService.updateCliente(c);
        }
        mav.addObject("resultado", key);
        mav.setViewName("Laboratorio/indexLaboratorio8");
    }
    return mav;
}
```

PROBANDO FUNCIONAMIENTO

Insertar nuevo cliente:

Figura 4 Nuevo cliente.

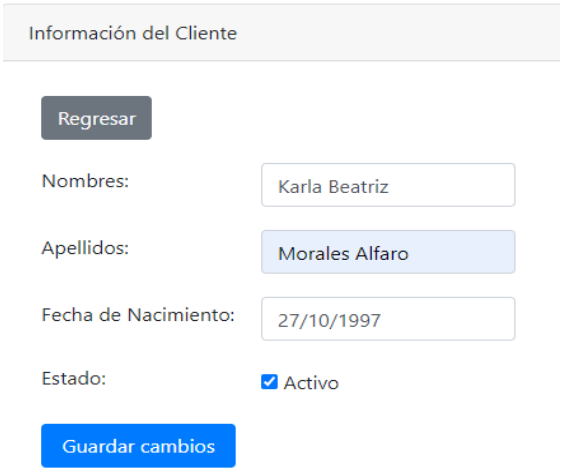
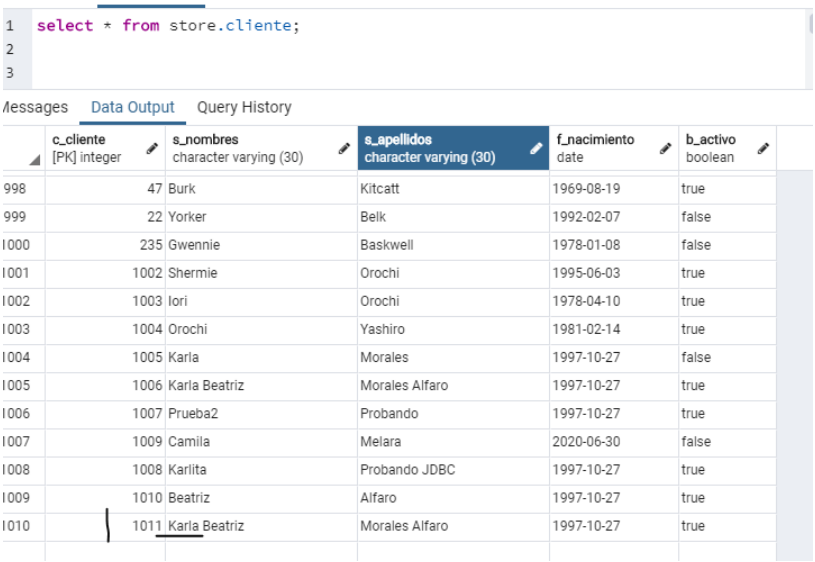


Figura 4.1 Guardar cambios y corroborar en la base de datos.



```
1  select * from store.cliente;
2
3
```

Messages | Data Output | Query History

	c_cliente [PK] integer	s_nombres character varying (30)	s_apellidos character varying (30)	f_nacimiento date	b_activo boolean
998	47	Burk	Kitcatt	1969-08-19	true
999	22	Yorker	Belk	1992-02-07	false
1000	235	Gwennie	Baskwell	1978-01-08	false
1001	1002	Shermie	Orochi	1995-06-03	true
1002	1003	Iori	Orochi	1978-04-10	true
1003	1004	Orochi	Yashiro	1981-02-14	true
1004	1005	Karla	Morales	1997-10-27	false
1005	1006	Karla Beatriz	Morales Alfaro	1997-10-27	true
1006	1007	Prueba2	Probando	1997-10-27	true
1007	1009	Camila	Melara	2020-06-30	false
1008	1008	Karlita	Probando JDBC	1997-10-27	true
1009	1010	Beatriz	Alfaro	1997-10-27	true
1010	1011	Karla Beatriz	Morales Alfaro	1997-10-27	true

Actualizar cliente:

Figura 4.2 Actualizar cliente.

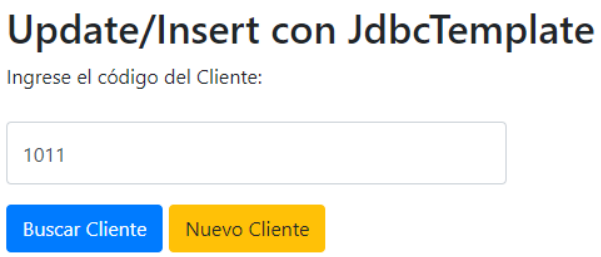
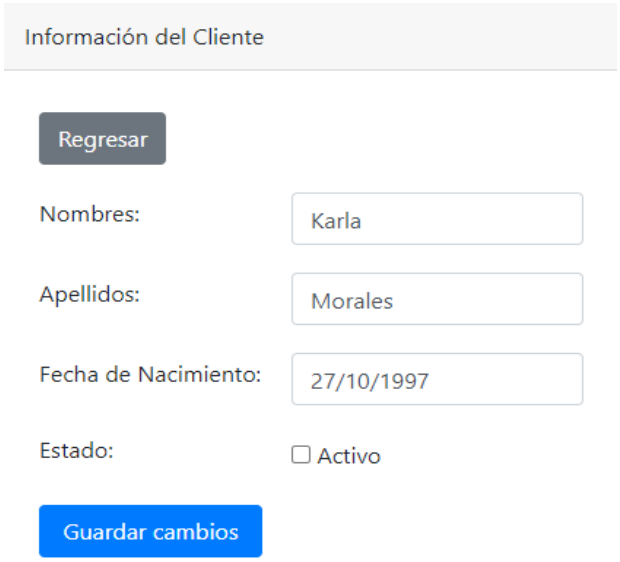
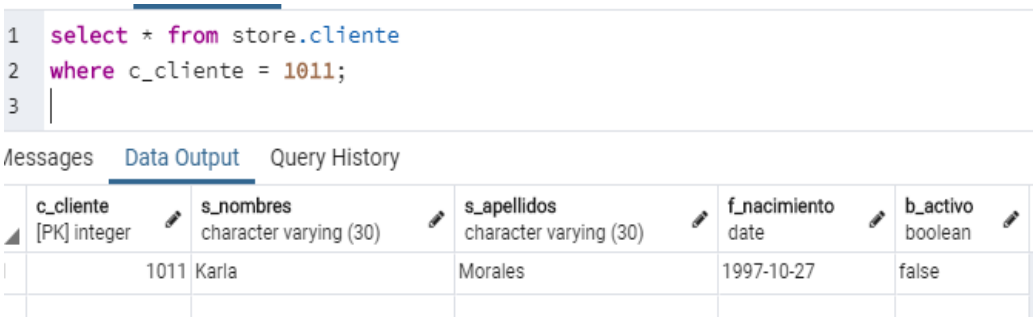


Figura 4.3 Guardar cambios y corroborar en la base de datos.





Procedimientos almacenados/funciones

----------CAPA DAO----------

Se hace uso de **SimpleJdbcCall.**

Figura 5 ClienteDAO.

```
public int ejecutarProcedimientoJdbc(Integer cliente, Boolean estado);
```

Figura 5.1 ClienteDAOImpl.

```
//PROCEDIMIENTO ALMACENADO
@Override
public int ejecutarProcedimientoJdbc(Integer cliente, Boolean estado) {
    SimpleJdbcCall jdbcCall = new SimpleJdbcCall(jdbcTemplate)
            .withSchemaName("store")
            .withProcedureName("sp_actualizar_cliente")
            .withoutProcedureColumnMetaDataAccess();

    //Se registran los parametros en el objeto correspondiente
    //Si es parametro de entrada -> new SqlParameter
    //Si es parametro de salida -> new SqlOutParameter
    jdbcCall.addDeclaredParameter(new SqlParameter("P_CLIENTE", Types.INTEGER));
    jdbcCall.addDeclaredParameter(new SqlParameter("P_ESTADO", Types.BOOLEAN));
    jdbcCall.addDeclaredParameter(new SqlOutParameter("P_SALIDA", Types.INTEGER));

    //Creamos un mapa con los nombres de los parametros y sus valores
    Map<String, Object> parametros = new HashMap<>();
    parametros.put("P_CLIENTE", cliente);
    parametros.put("P_ESTADO", estado);

    Map<String, Object> out = jdbcCall.execute(parametros);


    return Integer.parseInt(out.get("P_SALIDA").toString());

}
```

----------CAPA DE SERVICIO----------

Figura 6 ClienteService.

```
public int ejecutarProcJdbc(Integer cliente, Boolean estado);
```

Figura 6.1 ClienteServiceImpl.

```
@Override
public int ejecutarProcJdbc(Integer cliente, Boolean estado) {
    // TODO Auto-generated method stub
    return clienteDao.ejecutarProcedimientoJdbc(cliente, estado);
}
```

----------CAPA CONTROLADOR----------

Figura 7 Laboratorio8Controller.

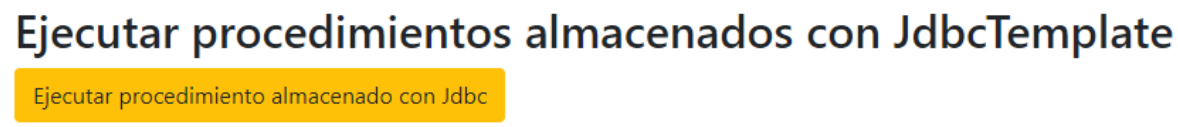
```
//PROCEDIMIENTO ALMACENADO
@RequestMapping("/procAlmacenadoJdbc")
public ModelAndView procAlmacenadoJdbc() {
    ModelAndView mav = new ModelAndView();
    mav.addObject("usuario", new Cliente());
    mav.setViewName("Laboratorio/procedimiento");
    return mav;
}
```

Figura 7.1 Laboratorio8Controller.

```
@RequestMapping("/ejecutarProcedimientoJdbc")
    public ModelAndView ejecutarProcedimiento(@RequestParam Integer cliente, @RequestParam Boolean estado){
        ModelAndView mav = new ModelAndView();
        Integer resultado;
        resultado = clienteService.ejecutarProcJdbc(cliente, estado);
        mav.addObject("resultado", resultado);
        mav.setViewName("Laboratorio/resultado");
        return mav;
}
```

PROBANDO FUNCIONAMIENTO

Figura 8 Ejecutar procedimiento almacenado.



Cambiar estado de un vehículo a partir del código cliente.

Figura 8.1 Cambiar estado de un vehículo y corroborar en la base de datos.

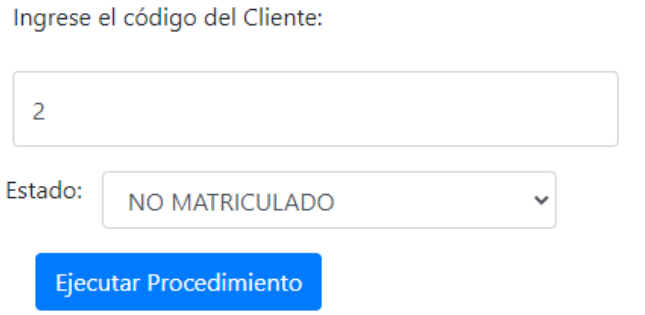


Figura 8.2 Cambiar estado de un vehículo y corroborar en la base de datos.

Figura 8.3 Cambiar estado de un vehículo y corroborar en la base de datos.

```
7
8  select * from store.vehiculo
9  where c_cliente= 2;
10
```

Messages | Data Output | Query History

	c_vehiculo [PK] integer	s_marca character varying (30)	s_modelo character varying (30)	s_chassis character varying (30)	f_compra date	b_estado boolean
1	6	Lincoln	Continental Mark VII	1G6KE54Y94U921836	2014-08-20	false
2	31	Buick	Regal	1FMJU1G50AE496057	2014-08-20	false
3	32	Chrysler	Town & Country	3D73M4HLXAG374657	2014-08-20	false
4	36	Chrysler	Town & Country	2G4GZ5GV6B9910890	2014-08-20	false
5	40	Nissan	Xterra	2G4WD552471567862	2014-08-20	false
6	42	Isuzu	Hombre Space	2T1BU4EE3CC389879	2014-08-20	false

Inserciones Batch

- JdbcTemplate permite ejecutar inserciones en “batch” (por lotes).
- Dichas inserciones se hacen cuando se quieren persistir una gran cantidad de registros.
- No hacerlo de esta manera, implicaría abrir y cerrar una conexión a la base de datos por cada registro (puesto que se haría un insert por registro).
- En cambio, realizarlo por lotes, permitiría hacer el insert de N registros en una sola conexión (un solo insert), mejorando drásticamente la velocidad

----------CAPA DAO----------

Figura 9 ClienteDAO.

```
public int[][] batchInsertVehiculos(final List<Vehiculo> vehiculos);
```

Figura 9.1 ClienteDAOImpl.

```
public int[][] batchInsertVehiculos(List<Vehiculo> vehiculos) {
    // TODO Auto-generated method stub
    String sql = "INSERT INTO store.vehiculo " +
            "(c_vehiculo, s_marca, s_modelo, s_chassis, f_compra, b_estado, c_cliente) VALUES (?,?,?,?,?,?,?)";

            int[][] resultado = jdbcTemplate.batchUpdate(sql, vehiculos, 1000, new ParameterizedPreparedStatementSetter<Vehiculo>() {

                @Override
                public void setValues(PreparedStatement ps, Vehiculo v) throws SQLException {
                    // TODO Auto-generated method stub
                    ps.setInt(1, v.getCvehiculo());
                    ps.setString(2, v.getSmarca());
                    ps.setString(3, v.getSmodelo());
                    ps.setString(4, v.getSchassis());
                    java.sql.Date fcompra = new java.sql.Date(v.getFcompra().getTime().getTime());
                    ps.setDate(5, fcompra);
                    ps.setBoolean(6, v.getBestado());
                    ps.setInt(7, v.getCcliente());
                }

            });

            return resultado;
}
```

----------CAPA DE SERVICIO----------

Figura 10 ClienteService.

```
public int[][] cargaMasiva() throws ParseException;
```

Figura 10.1 ClienteServiceImpl.

```
@Override
public int[][] cargaMasiva() throws ParseException {
    // TODO Auto-generated method stub
    List<Vehiculo> vehiculos = prepararColeccion();
    int[][] cantidad = clienteDao.batchInsertVehiculos(vehiculos);
    return cantidad;
}
```

En **prepararColeccion()** se procede a abrir el archivo vehiulos.csv.

- Asegurarse de poner correctamente la ruta absoluta del archivo.
- En este archivo se encontrarán los datos a insertar en la tabla vehículo.

Figura 10.2 ClienteServiceImpl prepararColeccion().

```
public List<Vehiculo> prepararColeccion() throws ParseException{
    String csv = "C:--RUTA ABSOLUTA\\vehiculos.csv"; //<----- ASEGURAR LA RUTA ABSOLUTA DONDE
                                                     //     ENCUENTRA EL ARCHIVO
    List<Vehiculo> coleccion = new ArrayList<Vehiculo>();
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
    Calendar cal = Calendar.getInstance();

    BufferedReader br = null;
    String line = "";
    String cvsSplitBy = ",";
    try {

        br = new BufferedReader(new FileReader(csv));
        while ((line = br.readLine()) != null) {

            String[] vStr = line.split(cvsSplitBy);
            Vehiculo v = new Vehiculo();
            v.setCvehiculo(Integer.parseInt(vStr[0]));
            v.setSmarca(vStr[1]);
            v.setSmodelo(vStr[2]);
            v.setSchassis(vStr[3]);
            cal.setTime(sdf.parse(vStr[4]));
            v.setFcompra(cal);
            v.setBestado(vStr[5].equals("t") ? true : false);
            v.setCcliente(Integer.parseInt(vStr[6]));

            coleccion.add(v);
        }

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (br != null) {
            try {
                br.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    return coleccion;
}
```

----------CAPA CONTROLADOR----------

Figura 11 Laboratorio8Controller.

```
//INSERCION BATCH

@RequestMapping("/batchVehiculo")
   public ModelAndView insercionBatch() throws ParseException{
       ModelAndView mav = new ModelAndView();
       long startTime = System.nanoTime();
       clienteService.cargaMasiva();
       long endTime = System.nanoTime();
       long duration = (endTime - startTime) / 1000000;
       logger.log(Level.INFO, "Duracion del metodo -> {0} milisegundos", duration);
       mav.setViewName("Laboratorio/resultado");
       return mav;
}
```

Antes de probar el funcionamiento se deben borrar los registros de la tabla vehículo, de lo contrario dará error de llaves duplicadas.

```
delete from store.vehiculo;
```

PROBANDO FUNCIONAMIENTO

Figura 12 Carga Masiva de vehículos.

	c_vehiculo [PK] integer	s_marca character varying (30)	s_modelo character varying (30)	s_chassis character varying (30)	f_compra date	b_estado boolean
1	1	Jeep	Grand Cherokee	7FJXMBLLJD	2014-08-20	true
2	3	Suzuki	Sidekick	JN8AF5MV4FT048599	2014-08-20	false
3	4	Dodge	Ram Wagon B350	JN8AZ1MU7EW554299	2014-08-20	false
4	5	Ford	Fiesta	5NPET4AC6AH133914	2014-08-20	false
5	7	BMW	7 Series	WAUGGBFC0DN380728	2014-08-20	false
6	8	Chevrolet	Express 2500	JTEBW3EHXA2854275	2014-08-20	true
7	9	Lincoln	Continental	3N1CN7AP2DL249169	2014-08-20	false
8	10	GMC	Sonoma Club Coupe	JTEBU4BF7AK663878	2014-08-20	true
9	11	Lincoln	Navigator	3C3CFFERXDT896603	2014-08-20	false
10	12	Infiniti	QX56	3VW4S7ATXEM277589	2014-08-20	false
11	13	Mercedes-Benz	CLS-Class	1FBNE3BL9BD087975	2014-08-20	false
12	14	Lexus	SC	1G6DP567450338204	2014-08-20	true
13	15	GMC	Sierra 3500	1FMCU0F76EU565299	2014-08-20	false

Todos los registros del archivo vehiculos.csv han sido insertados.

Tarea 100%:

Clonar el repositorio y seguir la guía exactamente, al final se deben tener las siguientes funcionalidades:

- Insertar/Actualizar.
- Ejecutar procedimiento almacenado.
- Hacer carga masiva (Batch).

Modo de calificación **(No habrá media nota)**:

- Funciona - **10/10.**
- No funciona: - **0/10**
    - No compila.
    - No funciona uno o más de uno de los 3 métodos.
    - Si no se hace uso de los métodos JDBC ejemplificados en la guía, la nota automáticamente será 0.

Fecha de entrega:

- Martes 30 junio 11:59pm.