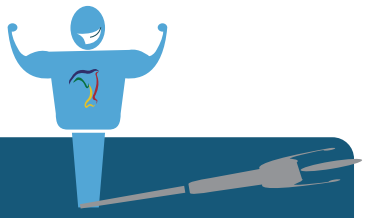
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

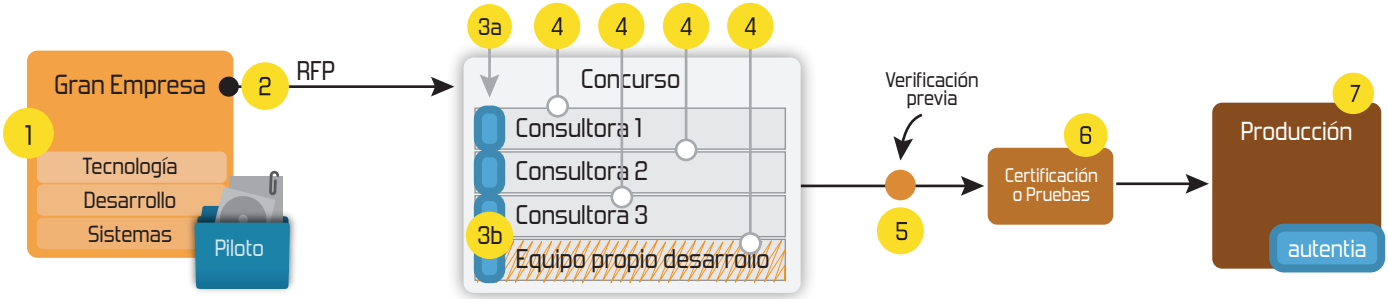
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por: Francisco Javier Martínez Páez

Puedes encontrarme en Autentia
Somos expertos en Java/J2EE
Contacta en info@autentia.com

autentia
real business solutions

Descargar este documento en formato PDF JDBCDesconocido.pdf

Firma en nuestro libro de Visitas

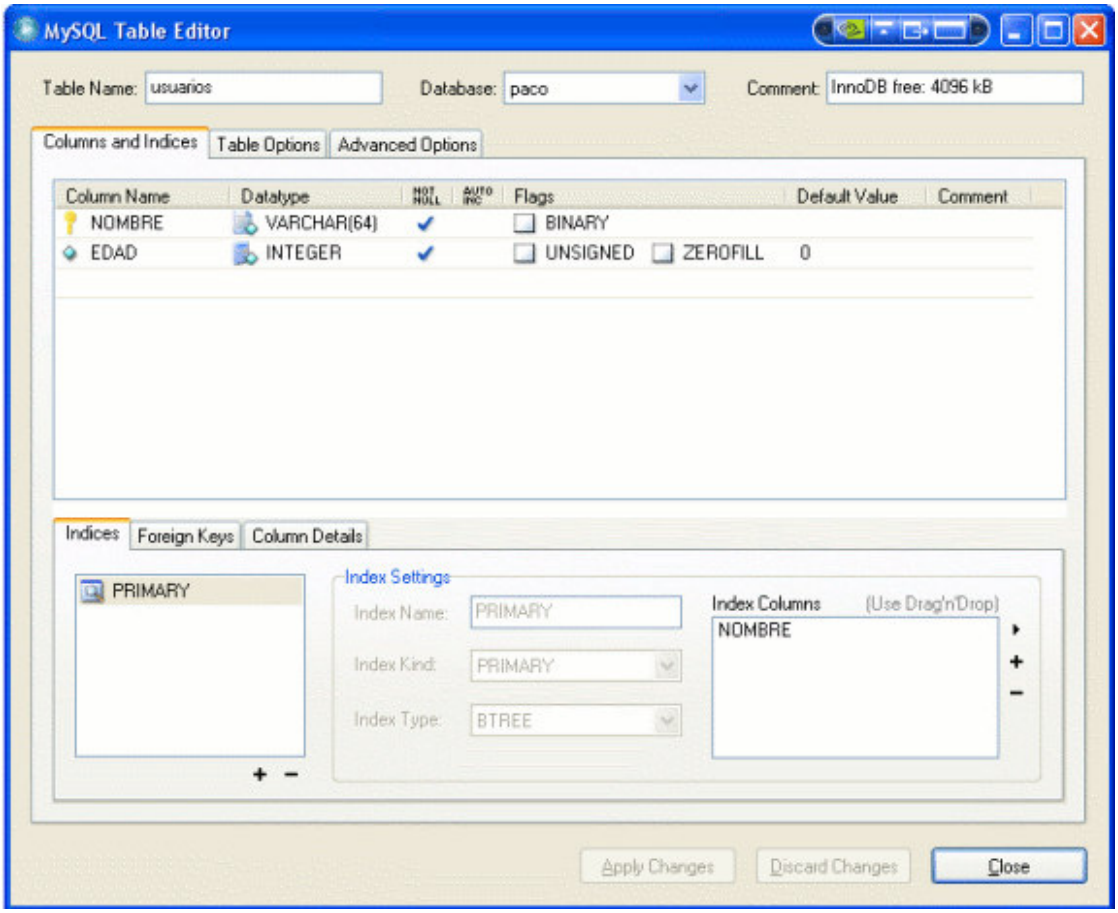
ALGUNAS CARACTERÍSTICAS MENOS CONOCIDAS DEL API JDBC.

Siguiendo un poco con los tutoriales sobre JDBC, en este pretendo poner de manifiesto, si el lector aun no las conoce, algunas posibilidades que nos permite el API de JDBC (y si el driver nos lo permite) que son menos conocidas.

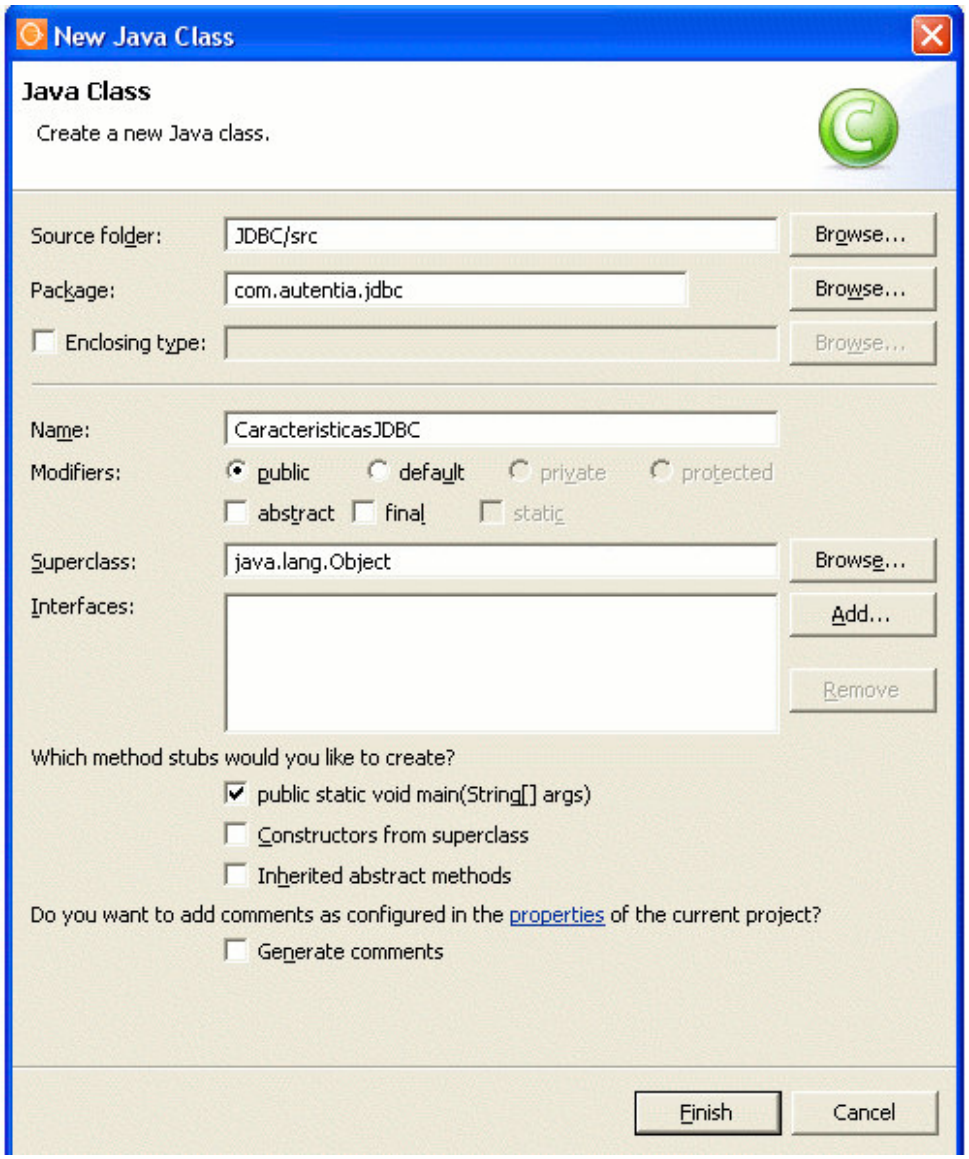
Los ejemplos de este tutorial están hechos con el siguiente entorno de desarrollo:

- Jboss Eclipse IDE Milestone 5.
- JDK 1.4
- MySQL 5.0
- MySQL Administrator (opcional)
- MySQL Connector/J (Driver tipo 4 que implementa la versión JDBC 3.0)

Para los ejemplos del tutorial usaremos la tabla USUARIOS:



Vamos a crear una nueva clase que denominaremos CaracteristicasJDBC:



Copiamos el método getConnection() de otros tutoriales y las constantes que usa:

```
protected static String dbClass = "com.mysql.jdbc.Driver";
protected static String dbUrl = "jdbc:mysql:///paco";

protected Connection getConnection() {

    Properties props = new Properties();
    props.put("user","<tu_usuario>");
    props.put("password","<tu_password>");
    Connection conBBDD = null;

    try {
        Class.forName(dbClass);
        conBBDD=DriverManager.getConnection(dbUrl, props);
    } catch(Exception e) {
         return null;
    }
    return conBBDD;
}
```

Ya podemos empezar:

Ejecutar sentencias por lotes.

Nos basaremos para el ejemplo en tres métodos de Statement:

- addBatch(String sentencia). Añade una sentencia a la lista de sentencias por lotes.

- executeBatch(). Ejecuta la lista de sentencias por lotes. Retorna un array de enteros con el resultado de cada sentencia.
- clearBatch(). Limpia la lista de sentencias por lotes.

Si tenemos la intención de realizar varias sentencias INSERT o UPDATE consecutivas, es mucho más eficiente utilizar la ejecución de sentencias por lotes que enviar las sentencias una por una.

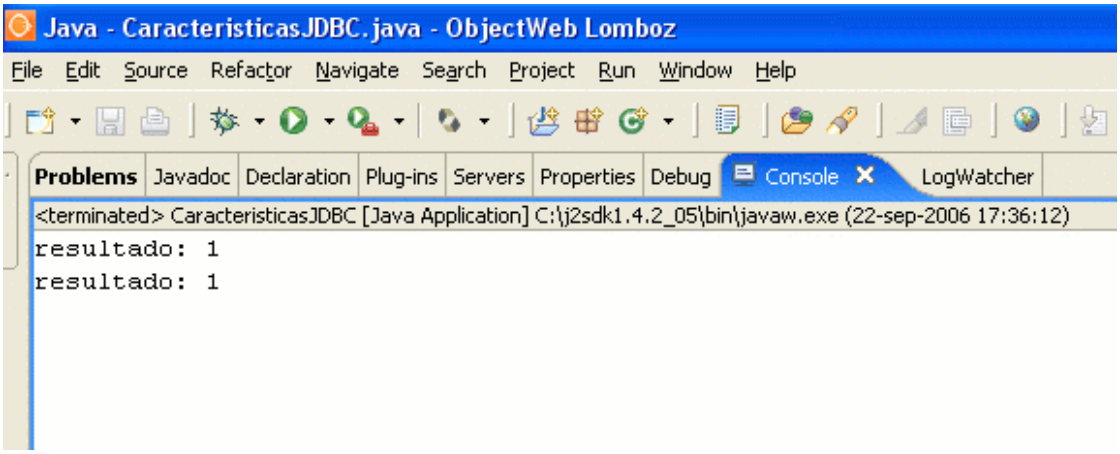
Creamos un método que reciba una lista de sentencias y las ejecuta por lotes:

```
public int[] ejecutaPorLotes(List sentencias) {
       Connection conn = getConnection();
       int[] resultado = null;
       Statement st = null;
       try {
          st = conn.createStatement();
          for (int i = 0; i < sentencias.size(); i++) {
             String sentencia = (String) sentencias.get(i);
             st.addBatch(sentencia);
          }
          resultado = st.executeBatch();
          st.clearBatch();
          return resultado;
       } catch (SQLException e) {
          e.printStackTrace();
          return resultado;
       } finally {
          if (st != null) {
             try {
                st.close();
             } catch (SQLException e) {
                e.printStackTrace();
             }
          }
          if (conn != null) {
             try {
                conn.close();
             } catch (SQLException e) {
                e.printStackTrace();
             }
          }

       }

   }
```

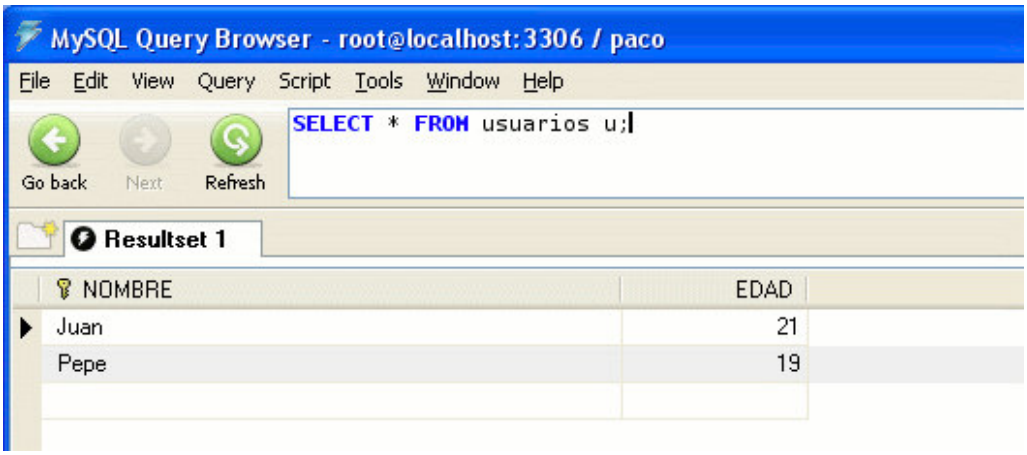
Vamos a probar el código:

```
public static void main(String[] args) {

       CaracteristicasJDBC car = new CaracteristicasJDBC();

       ArrayList lista = new ArrayList();
       lista.add("INSERT INTO USUARIOS VALUES ('Pepe',19)");
       lista.add("INSERT INTO USUARIOS VALUES ('Juan',21)");

       int[] resultado = car.ejecutaPorLotes(lista);

       for(int i=0;i<resultado.length;i++) {
          System.out.println("resultado: "+resultado[i]);
       }

   }
```

Ejecutamos y vemos la consola:



Comprobamos los datos en la tabla:



Sería buena idea también modificar el código para desactivar el modo autocommit (setAutoCommit(false)) en la conexión y realizar un rollback en caso de encontrarnos algún error en el array de enteros.

Utilizar un ResultSet Modificable.

En el siguiente ejemplo vamos a modificar el primero de los registros que introdujimos en el ejemplo anterior, usando algunos métodos de ResultSet.

El código del siguiente ejemplo lo vamos a realizar directamente en el método main y lo explicaremos en los comentarios del código:

```
public static void main(String[] args) {

      CaracteristicasJDBC car = new CaracteristicasJDBC();

      Connection conn = car.getConnection();

      Statement st = null;
      ResultSet rs = null;

      try {

         // Creamos un objeto de tipo Statement y le decimos al driver que vamos
        // a usar un ResultSet Scrollable y Updatable.
         st = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
              ResultSet.CONCUR_UPDATABLE);

         String query = "SELECT NOMBRE, EDAD FROM USUARIOS";

         // Obtenemos el ResultSet que debería contener dos registros.
         rs = st.executeQuery(query);

         // Nos situamos sobre el primero de ellos.
         rs.first();
```

```
            // Modificamos la columna EDAD del primer registro del ResultSet
            // (sólo en el ResultSet y no en la tabla)
            rs.updateInt("EDAD", 29);
            // Cancelamos el UPDATE que acabamos de realizar
            rs.cancelRowUpdates();
            // Volvemos a modificar la columna EDAD del primer registro del
            // ResultSet con el nuevo valor
            rs.updateInt("EDAD", 30);

            // Enviamos los cambios a la base de datos.
            rs.updateRow();

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (st != null) {
                try {
                    st.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }

        }

    }
```

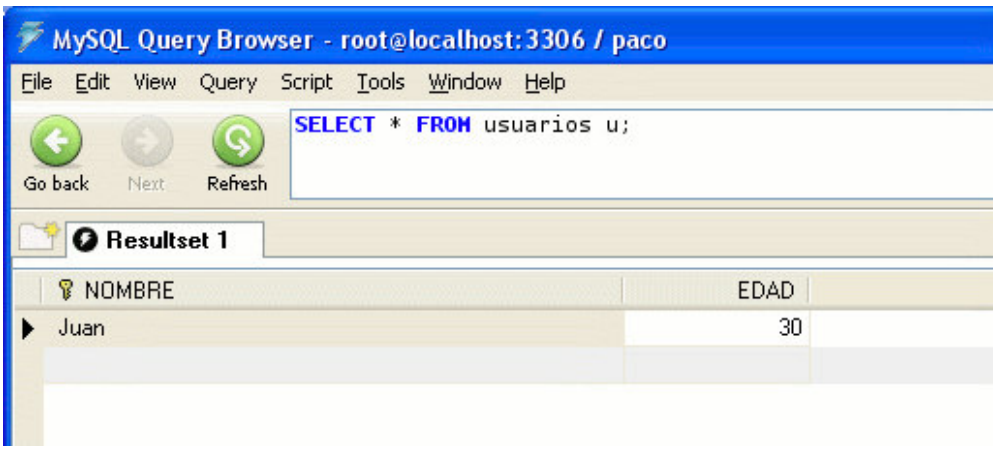
Una vez ejecutado el código, comprobamos el resultado en la tabla:

Vamos a modificar el código anterior para eliminar el último registro (Pepe)

```
public static void main(String[] args) {

        CaracteristicasJDBC car = new CaracteristicasJDBC();

        Connection conn = car.getConnection();

        Statement st = null;
        ResultSet rs = null;
```

```
        try {

        // Creamos un objeto de tipo Statement y le decimos al driver que vamos
        // a usar un ResultSet Scrollable y Updatable.
            st = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                 ResultSet.CONCUR_UPDATABLE);

            String query = "SELECT NOMBRE, EDAD FROM USUARIOS";

            // Obtenemos el ResultSet que debería contener dos registros.
            rs = st.executeQuery(query);

            // Nos situamos sobre el último de ellos.
            rs.last();

            // Eliminamos el registro actual.
            rs.deleteRow();

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (st != null) {
                try {
                    st.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }

        }

    }
```

Una vez ejecutado el código, comprobamos el resultado en la tabla:

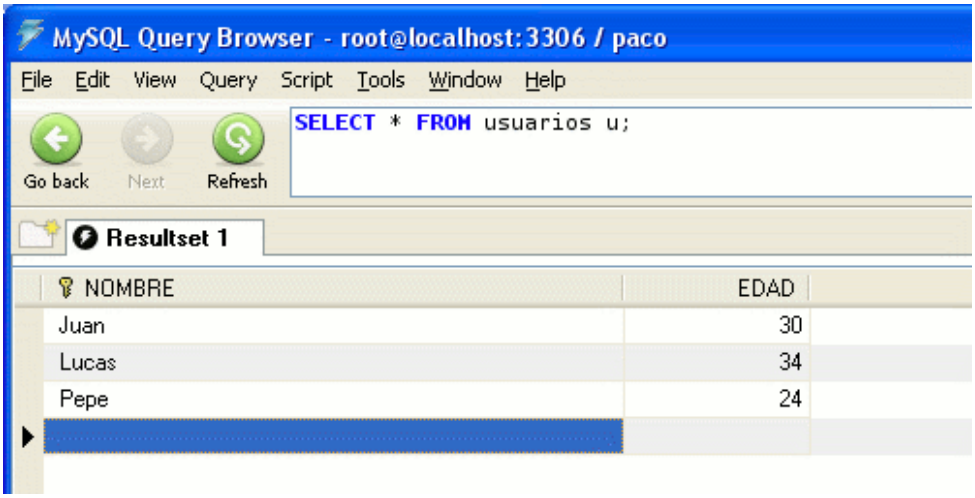


Vamos a modificar de nuevo el código anterior para insertar de nuevo a Pepe y a un amigo de Pepe que se llama Lucas.

```
public static void main(String[] args) {

        CaracteristicasJDBC car = new CaracteristicasJDBC();
```

```
        Connection conn = car.getConnection();

        Statement st = null;
        ResultSet rs = null;

        try {

            // Creamos un objeto de tipo Statement y le decimos al driver que
            // vamos a usar un ResultSet Scrollable y Updatable.
            st = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                  ResultSet.CONCUR_UPDATABLE);

            String query = "SELECT NOMBRE, EDAD FROM USUARIOS";

            // Obtenemos el ResultSet que debería contener dos registros.
            rs = st.executeQuery(query);

            // Nos situamos sobre el primero de ellos.
            rs.first();

            // Nos situamos sobre "insertRow". Esta fila no es más que un buffer
            // que nos permite construir un registro para insertarlo en la base
            // de datos.
            rs.moveToInsertRow();

            // Creamos el registro Lucas,34
            rs.updateString("NOMBRE", "Lucas");
            rs.updateInt("EDAD", 34);

            // Lo insertamos en la base de datos.
            rs.insertRow();

            // Creamos el resgistro Pepe,24
            rs.updateString("NOMBRE", "Pepe");
            rs.updateInt("EDAD", 24);

            // Lo insertamos en la base de datos.
            rs.insertRow();

            // Volvemos a situarnos sobre el registro en el que estábamos
            // previamente a invocar al método moveToInsertRow() (el primero)
            rs.moveToCurrentRow();

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (st != null) {
                try {
                    st.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }

        }

    }
```

Una vez ejecutado el código, comprobamos el resultado en la tabla:



Claves autogeneradas.

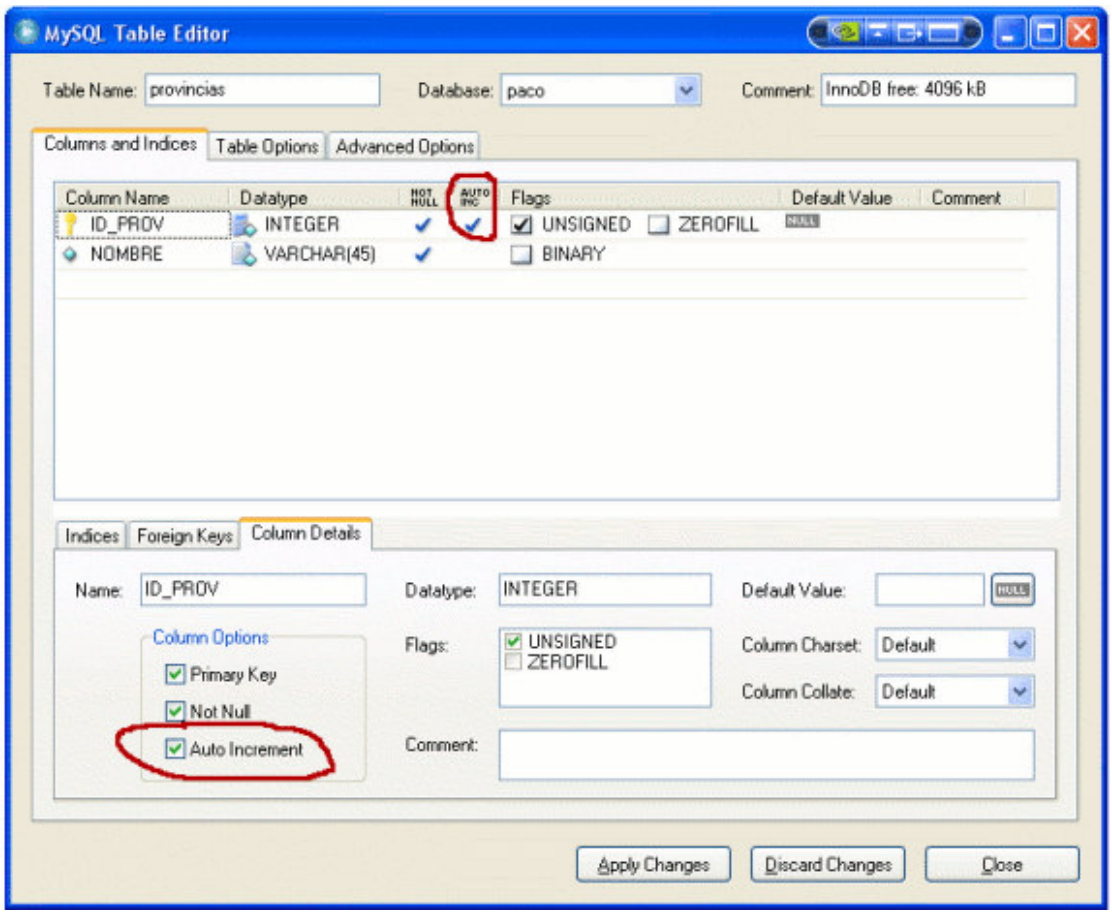
Algunos motores de base de datos (como mysql) permiten usar claves autogeneradas, es decir, como clave primaria de la tabla un campo autonumérico.

El problema viene cuando insertamos un registro en una tabla con claves autogeneradas y queremos saber cual es el valor de esa clave sin tener que hacer una consulta posterior. ¿ Es posible esto ? vamos a verlo.

Lo primero que vamos a hacer es crearnos una tabla llamada provincias con claves autogeneradas usando MySQLAdministrator:

La tabla tendrá dos campos:

- ID_PROV (Primary Key autogenerada)
- NOMBRE



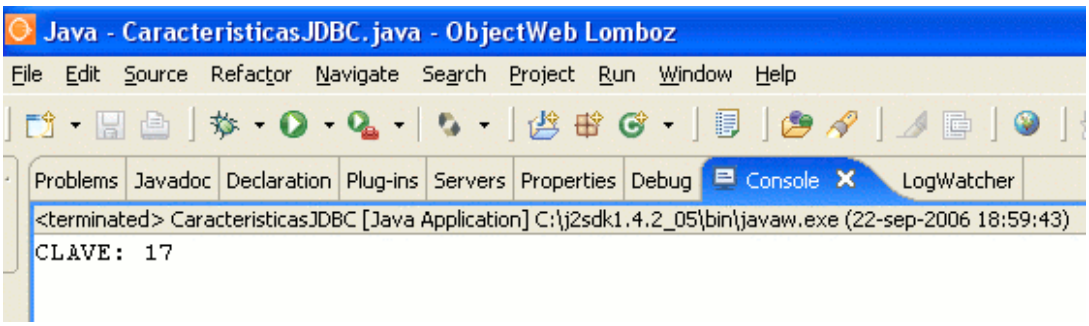
Para recuperar las claves autogeneradas haremos uso del método de Statement:

- getGeneratedKeys(): Devuelve un ResultSet donde se almacenan las claves creadas.

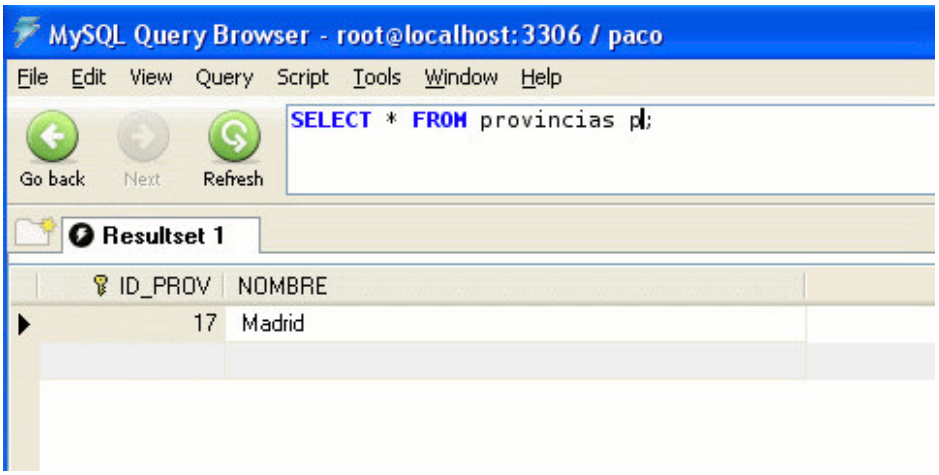
Vamos al ejemplo:

```
public static void main(String[] args) {

      CaracteristicasJDBC car = new CaracteristicasJDBC();

      Connection conn = car.getConnection();

      Statement st = null;
      ResultSet rs = null;

      try {

         st = conn.createStatement();

         // Insertamos un registro.
         st.executeUpdate("INSERT INTO PROVINCIAS (NOMBRE) VALUES ('Madrid')");

         // Obtenemos las claves autogeneradas y las mostramos por pantalla
         rs=st.getGeneratedKeys();

          while(rs.next()) {
             int id = rs.getInt(1);
             System.out.println("CLAVE: "+id);
          }

      } catch (SQLException e) {
         e.printStackTrace();
      } finally {
         if (rs != null) {
            try {
               rs.close();
            } catch (SQLException e) {
               e.printStackTrace();
            }
         }
         if (st != null) {
            try {
               st.close();
            } catch (SQLException e) {
               e.printStackTrace();
            }
         }
         if (conn != null) {
            try {
               conn.close();
            } catch (SQLException e) {
               e.printStackTrace();
            }
         }

      }

   }
```

Ejecutamos el código y mostramos la consola:



Vamos a ver los registros en la tabla:



Puntos intermedios de rollback.
JDBC permite manejar la transaccionalidad desactivando el modo autocommit (autoentrega) con el método de Connection: setAutoCommit(false). Todos conocemos además los métodos:

- commit(). Realiza los cambios
- rollback(). Deshace los cambios

A partir de la versión jdbc 3.0 se incluye una nueva posibilidad: crear puntos de salvaguarda o SavePoint para hacer rollback parciales:

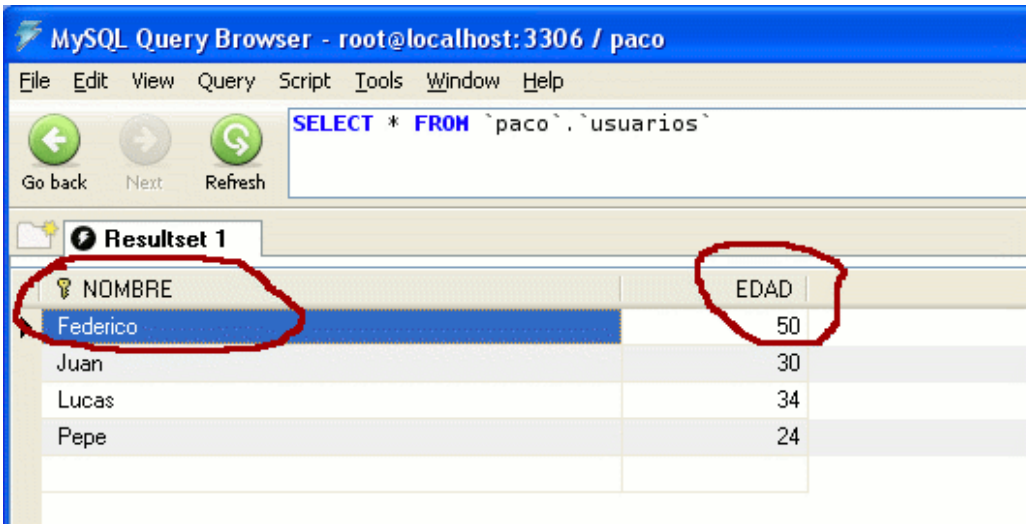
```
SavePoint sp = conn.setSavePoint("NOMBRE");
....
conn.rollback(sp);
```

Vamos a hacer un ejemplo y comentaremos en el código lo interesante:

```
public static void main(String[] args) {

    CaracteristicasJDBC car = new CaracteristicasJDBC();

    Connection conn = car.getConnection();

    Statement st = null;
    ResultSet rs = null;

    try {
       // desactivamos el método de autoentrega.
       conn.setAutoCommit(false);

       st = conn.createStatement();

       // Insertamos un registro en la tabla de usuarios
       st.executeUpdate("INSERT INTO USUARIOS  VALUES ('Federico',50)");

       // Creamos un punto de salvaguarda en este momento.
       Savepoint sp = conn.setSavepoint("PUNTO1");

       // Modificamos la edad de Federico:
       st.executeUpdate("UPDATE USUARIOS SET EDAD = 45 WHERE          NOMBRE = 'Federico'");

       // Hacemos un rollback al punto PUNTO1:
       conn.rollback(sp);

       // Entregamos los cambios anteriores a punto1 a la BBDD:
       conn.commit();


    } catch (SQLException e) {
       e.printStackTrace();
    } finally {
       if (rs != null) {
          try {
             rs.close();
          } catch (SQLException e) {
             e.printStackTrace();
          }
       }
```

```
            if (st != null) {
                try {
                    st.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }

        }

    }
```

Si ejecutamos el código anterior, deberíamos insertar un nuevo registro en la tabla de usuarios con nombre Federico y con edad 50, ya que el cambio posterior lo hemos anulado. Una vez ejecutado comprobamos la tabla:



Bueno, pues no ha sido demasiado dificil.

Lo de siempre, si quereis ayuda no tenéis mas que poneros en contacto con nosotros: http://www.autentia.com

Puedes opinar sobre este tutorial aquí

Recuerda

que el personal de Autentia te regala la mayoría del conocimiento aquí compartido (Ver todos los tutoriales)

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?

¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?

info@autentia.com

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos .....
Autentia = Soporte a Desarrollo & Formación

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	**Descripción**
Introducción a JDBC	En este tutorial os explicamos los fundamentos teóricos de JDBC
CMP Entity Beans y MySql	Os mostramos como crear un Entity Bean con persistencia controlada por el servidor, configurado para usar MySql
Administracion Web de MySQL	Os mostramos como instalar en vuestro PC phpMySQL, un potente gestor Web de MySQL utilizado en la mayoría de los Hostings.
Modelado Gráfico de MySQL	Os mostramos como instalar y utilizar DeZing e Importer de Datanamic para crear el modelo lógico y físico de bustra base de datos MySQL
Apache, MySQL y PHP	Os mostramos como configurar Apache, MySQL y PHP en vuestra máquina
Modelado de MySQL con herramientas gratuitas	Os mostramos otra alternativa de modelado gráfico de MySql.
Generación automática de código JDBC	En este tutorial os enseñamos como, sin conocimiento de JDBC, crear vuestro programas en Java, gracias a JDBCTest.
MySql en Windows	MySql es una de las principales bases de datos "gratuitas" que podemos encontrar en Internet. En este tutorial aprendereis a instalarlo en Windows
CachedRowSet: JDBC y Java 5.	En este tutorial aprenderemos a el uso y funcionamiento del rowSet de tipo CachedRowSet.
JDBC y MySql	En el tutorial anterior vimos como instalar MySQL en Windows, ahora vamos a ver como acceder desde una aplicación Java.