Desarrollo de Software
3º semestre

Programa de la asignatura:
Programación orientada a objetos II

Unidad 3. Base de datos

Ciudad de México, abril del 2025

Clave:
Ingeniería: **TSU:**
15142421 / 16142421

Universidad Abierta y a Distancia de México

Índice

Unidad 3. Base de datos

Presentación de la Unidad

Bienvenido a la tercera y última unidad de POO2, en donde se abordará el tema de base de datos. Este tema te ayudará a que conozcas y comprendas cómo conectar un programa a una base de datos.

Además de ello, se realizará un repaso sobre el lenguaje DML (lenguaje de manipulación de datos, por sus siglas en inglés) y DDL (lenguaje de definición de datos, por sus siglas en inglés) para que recuerdes sentencias SQL; una vez que hayas adquirido este conocimiento, podrás hacer programas que realicen conexiones a bases de datos y manipular su contenido.

En esta unidad recordarás sentencias SQL, que viste en materias de bases de datos, utilizarás tus conocimientos sobre interfaces gráficas, eventos y podrás darles uso para manipular información desde un programa.

Recuerda que, si te surgen dudas respecto al contenido de la unidad y los temas mencionados, puedes consultarlas con tu Docente en línea para una mejor comprensión de los temas.

Logros

Al término de esta unidad lograrás:

- Identificar las sentencias del DML y DDL.
- Distinguir los componentes que conforman una conexión a base de datos.
- Emplear la conexión a base de datos y mostrar los resultados.

Competencia específica

- Utilizar las características de la programación orientada a objetos y de la creación de módulos para implementar aplicaciones ligadas a las bases de datos mediante la cadena de conexión, consultas y la presentación de resultados.

3.1. Lenguaje estructurado de consulta (SQL)

Este tema tiene la intención de recordar el lenguaje SQL para la creación y manipulación de bases de datos; el tema se revisa con más detalle en la asignatura Bases de datos.

¿Conoces SQL?
SQL es el lenguaje de consultas a bases de datos más ampliamente utilizado. Este lenguaje (SQL), debido a su uso constante, se convirtió en estándar, para hacer compatibles las sentencias de manejo de bases de datos, prácticamente todos los manejadores de bases de datos trabajan con este lenguaje, con algunas ligeras variaciones entre gestores (De Miguel, 1999).

Este lenguaje, al igual que cualquier lenguaje de programación, ha tenido variaciones a lo largo de su existencia, para irse adaptando a las necesidades de la industria del software y bases de datos.

Observa un ejemplo de un SQL muy básico.

SELECT *all* FROM *tabla.basededatos*;

Ahora que ya es posible identificarlo, se puede conocer más en los siguientes subtemas que consisten en: crear bases de datos mediante sentencias SQL (DDL) y manipular bases de datos (DML).

3.1.1. Lenguaje de definición de datos (DDL)

El lenguaje SQL puede dividirse entre aquellas sentencias (conjunto de palabras reservadas que unidas funcionan como una instrucción) que definen o crean bases de datos; aquellas que manipulan las bases de datos ya existentes y quien las administran.

En orden lógico, lo primero que se tratará es el lenguaje de definición de datos llamado DDL por sus siglas en inglés (Data Definition Language, en español lenguaje de definición de datos), este lenguaje permite la definición de las estructuras que conformarán las bases de datos.

La primera sentencia que se verá es la de crear bases de datos, observa el siguiente ejemplo:

```
CREATE SCHEMA `nombreBD`;
```

En el ejemplo el esquema es llamado *nombreBD*; visiblemente al crear otra base de datos la sentencia queda igual, a excepción del nombre que se vaya a colocar a la base de datos.

Una vez ejecutada la sentencia anterior, ya existe la base de datos, pero sin nada dentro (queda sólo como un cascarón), por lo tanto, se debe crear su estructura, añadiendo tablas, observa el siguiente ejemplo:

```
CREATE  TABLE `nombrebd`.`nombretabla` (

  `llavePrimaria` INT NOT NULL ,

  `campo1` VARCHAR(45) NULL ,

  `campo2` VARCHAR(45) NULL ,

  PRIMARY KEY (`llavePrimaria`) );
```

Para crear la tabla se comienza indicando CREATE TABLE, después se coloca el nombre de la base de datos donde se colocara la tabla y el nombre de la tabla que se creará, `*nombrebd*`.`*nombretabla*`; en seguida, entre paréntesis, se colocan los campos de los que estará compuesta dicha tabla, esto se hace colocando el nombre del campo (recuerda que un campo es donde se almacena un sólo dato; es la unidad más pequeña de almacenamiento de base de datos), su tipo de dato y la indicación de se acepta o no valores nulos, es decir si aceptará que al llegar un registro se omita ese campo `*campo1*` *VARCHAR(45) NULL*, se colocan seguidos todos los campos separados por una coma.

Nota: Se pueden seguir creando más tablas para completar el proceso de creación de una base de datos.

Las otras sentencias de las que está compuesto el DDL son aquellas que eliminan bases de datos o realizan algún cambio sobre ellas, si se quisiera eliminar la base de datos completa se emplearía la sentencia DROP, de la siguiente manera:

```
DROP SCHEMA `nombrebd`
```

Si se quiere eliminar sólo una tabla en especial, es de la siguiente manera:

```
DROP TABLE `nombrebd`.`nombretabla`
```

Por otra parte, si no quiere eliminarse, sino sólo realizar algún cambio, se realiza con la sentencia ALTER, para cambiar el nombre de una tabla, sería como en el ejemplo siguiente:

```
ALTER TABLE `nombrebd`.`nombretabla` RENAME TO  `nombrebd`.`nombre`
;
```

Si se quiere cambiar algún campo en especial:

```
ALTER TABLE `nombrebd`.`nombretabla` CHANGE COLUMN `campo2` `campo`
VARCHAR(45) NULL DEFAULT NULL  , RENAME TO  `nombrebd`.`nombre` ;
```

En este primer subtema has visto cómo crear bases de datos mediante sentencias SQL predefinidas para este propósito llamadas sentencias DD. Este subtema fue creado para que recuerdes las sentencias que ya has visto en la materia de Base de datos, ya que serán útiles para completar esta materia y que logres crear programas que empleen bases de datos.

3.1.2. Lenguaje de manipulación de datos (DML)

Como su nombre lo indica DML (Data Manipulation Languaje, en español lenguaje de manipulación de datos) está compuesto por aquellas sentencias y cláusulas que manipulan la información de las bases de datos, es decir, insertan, seleccionan, modifican o eliminan registros de la base de datos, sin realizar ningún tipo de alteración sobre la estructura de la base de datos, afectando sólo la información contenida en ella (Ceballos, 2006).

Observa la escritura de datos, esto se hace mediante la sentencia INSERT, la estructura de esta instrucción es la siguiente:

```
INSERT INTO nombreBD.tablaBD VALUES ('valorCampo1','valorCampo2',
'valorCampo3');
```

Ya con los datos insertados, si se requiere hacer una modificación sobre el registro se hace mediante la sentencia UPDATE, esta sentencia requiere colocar el nombre del campo que se va a modificar con su nuevo valor, indicando cuál de todos los registros se cambiará mediante la cláusula WHERE, donde se indica qué campo es el que se toma como referencia para encontrar el registro a cambiar, como se muestra en el siguiente ejemplo:

```
UPDATE nombreBD.tablaBD SET campoCambiar='valor' WHERE
campoComparar='valor';
```

Si lo que se requiere es eliminar un registro, esto se hace mediante la sentencia DELETE, se coloca el nombre de la tabla y mediante la cláusula WHERE se indica el valor a buscar para elegir el registro que se borrara, como se muestra en el siguiente ejemplo:

```
DELETE FROM nombreBD.tablaBD WHERE campoComparar='valor';
```

Por último, se verá la sentencia SELECT, que es de las más utilizadas, y con ella se pueden seleccionar todos los registros de una tabla:

```
SELECT * FROM nombreBD.tablaBD;
```

Seleccionar todos los registros de una tabla y mostrarlos ordenados en base al contenido de un campo en particular:

```
SELECT * FROM nombreBD.tablaBD ORDER BY campo;
```

Seleccionar sólo el contenido de un registro en una tabla en base al valor que contenga un campo en particular:

```
SELECT campo2  FROM nombreBD.tablaBD WHERE campo1='valor';
```

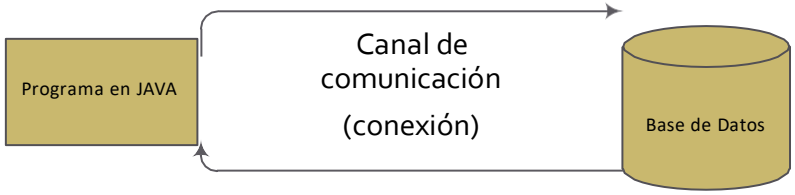
Estos temas sobre sentencias y cláusulas de SQL, cuentan con una gran variedad de combinaciones y posibilidades, pero por el momento sólo se muestran las construcciones básicas con el objetivo de que las recuerdes y puedas utilizarlas para construir un programa que cree y manipule bases de datos.

3.2. Conexión a base de datos

En el subtema anterior se repasaron sentencias SQL tanto para crear bases de datos, como para manipular la información dentro de ellas, en este tema se verá cómo conectar un programa a una base de datos, ya que si no se conectan (la base de datos y el programa) no se pueden utilizar las sentencias SQL para manipular información.

Una vez que se tiene una conexión entre un programa y una base de datos se pueden utilizar sentencias SQL (como las que se repasaron en el tema anterior) y lograr crear un programa que pueda manipular información.

Para lograr conectar un programa a una base de datos, primero se debe crear un canal de comunicación entre ambos para que por medio de este canal la información viaje. Imagina el canal de comunicación como si fuese un túnel entre el programa y la base de datos que sirve exclusivamente para que el programa y la base de datos intercambien información. Para que sea más claro observa la siguiente imagen donde se ejemplifica la conexión a BD.



Ejemplificación de conexión a BD.

El canal de comunicación es creado mediante un conector; este conector es una librería creada por el fabricante del manejador de base de datos que se vaya a utilizar. En este caso se utilizará el gestor de base de datos MySql, por lo que el conector que se utilizará es el Connector/J 5.1, (que es la última versión liberada a la fecha para conectar una base de datos de MySQL y un programa Java, si encuentras una versión más avanzada puedes utilizarla).

> Nota: Esta librería puede ser descargada de manera gratuita de la página oficial de MySQL. Oracle, Download Connector. Recuperado de http://dev.mysql.com/downloads/connector/j/
> En esta página encontrarás disponible la descarga del conector.

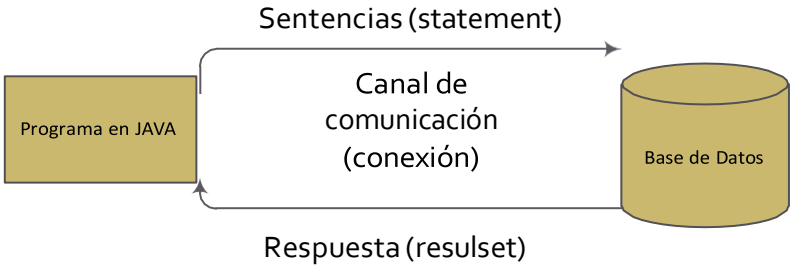
Es importante tener el conocimiento de que se utiliza este conector (Connector/J) y no otros de los disponibles, porque éste es el creado especialmente para Java, ya que existen conectores también para otros lenguajes. Lo que se debe hacer con este conector es descargarlo, descomprimirlo y añadir el archivo llamado mysql-connector-java-5.1. -**bin.jar** dentro de la carpeta del proyecto de NetBeans desde el que se esté haciendo la conexión, para así tener el conector y agregarlo a las clases a utilizar en el respectivo proyecto.

Una vez creado el canal de comunicación (mediante un conector), desde el programa hacia la base de datos viajan sentencias SQL, que crearán o manipularán la base de datos, esto

debe ir en un Statement, este es un objeto de tipo declaración estandarizado por Java para el envío de sentencias.

Para que el programa reciba las respuestas que la base de datos otorga al programa, también existe un objeto de tipo tabla estandarizada por Java, llamado ResultSet, en este objeto se contendrán los resultados de la sentencia que se realice, se ajustará precisamente a los resultados para acomodarlos en filas y columnas (como una tabla).

En los siguientes puntos se verá la declaración y uso de la conexión a base de datos. El gráfico con la representación de esta conexión se encuentra en la siguiente figura.



Ejemplificación del viaje de información a una BD.

En conclusión, para que un programa se comunique con una base de datos lo primero que se debe realizar es crear una conexión entre ambos (programa y base de datos), para ello se requiere un conector, el cual varía dependiendo del lenguaje de programación (en este caso Java) y del gestor de base de datos a utilizar (en este caso MySQL).

3.2.1. Cadena de conexión

Se debe recordar que para conectar una base de datos a un programa se utiliza un conector, para que ese conector realice su función (conectar la BD y el programa) debe recibir una cadena la de conexión, que es lo que a continuación se tratará. Esta cadena de conexión contiene la información que el conector requiere para realizar su trabajo.

Retomando el subtema anterior, lo primero que se debe hacer es descargar el conector, descomprimirlo y agregar el archivo a la carpeta del proyecto sobre el que se realizará la conexión a base de datos.

Una vez agregada la librería para la conexión, en la carpeta del proyecto del programa que se esté realizando se debe crear dicha conexión, como se muestra en el ejemplo de código.

- La primera línea (en morado) carga los archivos que crean la conexión desde el conector, como parámetro se envía el nombre del paquete donde se encuentran las clases y métodos que crean la conexión entre Java y el manejador de base de datos MySql (esta línea sólo cambia si se utiliza un conector diferente).
- La segunda línea (en azul) crea la conexión, lo primero es crear un objeto de la clase Connection (en este ejemplo llamado *conexion*), es en este objeto donde se crea la cadena de conexión mediante el paso de tres parámetros separados por coma, a continuación se desglosa la explicación de los parámetros solicitados:
    - El primer parámetro es la ubicación de la base de datos a utilizar, *jdbc:mysql://localhost/ejemploBD*, esto tiene tres partes.
        - La primer parte es igual, siempre que utilices el manejador MySql (*jdbc:mysql:*).
        - La segunda parte es la ubicación de la base de datos, en el ejemplo mostrado el acceso a la base de datos es de manera local, es decir la base de datos está en el mismo equipo que el programa, para los casos en que la conexión es remota, en lugar de *localhost*, se coloca la IP de la dirección de destino.
        - Y la última parte de este parámetro es el nombre de la base de datos con la que el programa se va a conectar (*ejemploBD*).
    - El siguiente parámetro es el nombre de usuario para acceder a la base de datos (este nombre es el que se configura al instalar el manejador de bases de datos).
    - Por último se coloca la contraseña de acceso a la base de datos (éste también se configura al instalar el manejador de bases de datos).

Ahora observa cómo quedaría en el siguiente ejemplo de código de cadena de conexión a BD:

```
Class.forName("org.gjt.mm.mysql.Driver");
            Connection                    conexion                    =
DriverManager.getConnection("jdbc:mysql://localhost/ejemplo",
"usuario", "contraseña");
```

Como se ha podido ver en el subtema, la creación de la cadena de conexión a bases de datos sólo requiere que se cree un objeto de conexión (de la clase Connection) y se le brinden los parámetros de localización, usuario y contraseña de la base de datos. Una vez que se logre crear esta conexión ya estará habilitada la comunicación entre el programa y la base de datos, y se comenzará con el intercambio de información entre ellos, también se podrá manipular información de una base de datos desde un programa, esto es parte de siguiente tema.

3.2.2. Consultas

Una consulta es una instrucción mediante la cual una base de datos proporciona información respecto a los datos que contiene (la base de datos). En este subtema se verá cómo enviar consultas en lenguaje SQL hacia una base de datos desde un programa, para obtener y/o manipular información.

Para poder crear un flujo de información entre un programa y una base de datos es importante retomar los subtemas anteriores, en los que se recordó el lenguaje creado para este propósito (3.1. Lenguaje estructurado de consulta (SQL)) y también se repasaron los tipos de consultas que se pueden hacer sobre las bases de datos (3.1.1. Lenguaje de definición de datos (DDL) y 3.1.2. Lenguaje de manipulación de datos (DML)). Asimismo, se debe tener presente el subtema de cadena de conexión, que como se verá más adelante se utilizará una cadena de conexión para poder enviar consultas a la base de datos y obtener información.

A continuación se verá lo que se necesita para crear un programa que envíe sentencias SQL a una base de datos:

- Lo primero es descargar el conector que creará la comunicación entre el programa y la base de datos.
- Agregar el conector a la carpeta del proyecto del programa que se va a crear.
- Generar la conexión entre la base de datos y el programa mediante la cadena de conexión (subtema anterior).
- Enviar sentencias SQL a la base de datos.

Una vez establecida la conexión a la base de datos ya se pueden enviar sentencias SQL a la base de datos, esto se hace mediante la creación de un objeto que Java creó específicamente para este propósito, llamado Statement, este objeto envía sentencias SQL a la base de datos a la que se ha conectado.

El objeto Statement se crea a partir del objeto de la conexión previamente hecha de manera que un Statement se crea mediante las siguientes líneas:

- Para crear un objeto Statement primero se debe tener ya creada una conexión (se retoma el ejemplo del subtema anterior en rosa),
- En seguida se crea el Statement con base en la conexión previamente realizada (azul).
- Ya con el Statement creado, se elabora una cadena que contendrá la sentencia SQL que se requiere ejecutar (morado).
- Por último se ejecuta la sentencia mediante el Statement creado (verde), para esto existen varios métodos, pero sólo se utiliza uno a la vez dependiendo del tipo de

consulta a ejecutar, se muestran los siguientes métodos del objeto Statement, mediante los que se ejecutan las sentencias (ORACLE, 2012):

- *execute(consulta)*: Ejecuta una sentencia de SQL que puede devolver varios resultados.
- *executeBatch()*: Envía un lote de comandos a la base de datos para su ejecución y si todos los comandos se ejecutan con éxito, devuelve un array de cuentas de actualización.
- *executeQuery(consulta)*: Ejecuta una sentencia SQL, que devuelve un sólo objeto ResultSet.
- *executeUpdate(consulta)*: Ejecuta una sentencia SQL, que puede ser un INSERT, UPDATE o DELETE, o una sentencia SQL que no devuelve nada, como una sentencia de SQL DDL.

Examina punto por punto el siguiente ejemplo de código de ejecución de consultas para comprender mejor la sintaxis previamente descrita.

Ejemplo de código de ejecución de consultas:

```
Class.forName("org.gjt.mm.mysql.Driver");
            Connection conexion =
DriverManager.getConnection("jdbc:mysql://localhost/ejemplo",
"usuario", "contraseña");

Statement st = conexion.createStatement();
String consulta="SELECT * FROM ejemploBD;

st.execute(con);
st.executeBatch();
st.executeQuery(con);
st.executeUpdate(insercion);
```

Para comprender mejor todos los temas tratados hasta el momento (conexión a BD, cadena de conexión, consultas y también DDL y DML) se ha creado un programa de ejemplo para el envío de consultas, el cual crea la conexión entre un programa y una base de datos, crea una cadena de conexión, envía información hacia la base de datos y recibe respuesta de la base de datos. El programa de ejemplo para el envío de consultas se describirá en los siguientes párrafos, el código se encuentra en el script 1, así como en el ejemplo de código 3, 4 y 5 este programa de ejemplo para el envío de consultas cuenta con tres clases, la primera llamada **Inicio** (observa el ejemplo de código 3**)** es la que contiene el método **main**, para inicializar el programa, la segunda clase llamada **Interfaz** (observa el ejemplo de código 4**)**, que crea la apariencia gráfica del ejemplo, y la tercera clase llamada **Conexión** (observa el ejemplo de código 5**)** es donde se crea la conexión y cadena de conexión a la base de datos, ejecuta sentencias y recibe respuesta de la base de datos. Además de las clases también se presenta el script para que construyas la base de datos.

En las clases mencionadas que contiene el ejemplo para el envío de consultas se utilizará una base de datos llamada ejemplo, con una tabla llamada usuario, la cual se crea mediante el script 1, ésta contiene dos campos **idusuario** y contra, los cuales son de tipo **varchar**.

A continuación se muestra el script con el que podrás generar la base de datos a utilizar. En este script se:

- Crea la base de datos `CREATE SCHEMA `ejemplo` ;`
- Se crea la tabla que contiene la base de datos `CREATE TABLE `ejemplo`.`usuario``
- Se crean también los campos que conformarán la tabla
  - `` `idusuario` VARCHAR(10) NOT NULL ``
  - `` `contra` VARCHAR(20) NOT NULL ``
- Y por último se define como llave primaria el campo idusuario, `PRIMARY KEY (`idusuario`)`

Recuerda que las llaves primarias en las tablas de bases de datos indican que ese campo no puede tener el mismo dato repetido; se eligió el idusuario como llave primaria para evitar que existan usuarios con el mismo nombre, a continuación, se muestra el script que genera la base de datos:

```
CREATE SCHEMA `ejemplo` ;

CREATE  TABLE `ejemplo`.`usuario` (

  `idusuario` VARCHAR(10) NOT NULL ,

  `contra` VARCHAR(20) NOT NULL ,

  PRIMARY KEY (`idusuario`) );
```

Script para la creación de una BD

Como se explicó anteriormente, el ejemplo para el envió de consultas consta de tres clases (ejemplo de código 3, 4 y 5): primero se presentará la clase main (ejemplo de código 3.), que es la que inicia el programa (recuerda que siempre debe existir una clase inicializadora que tenga un método main, pues así lo pide Java).

La clase inicio, cuenta con las dos siguientes secciones de código:

- La sección marcada en azul es la declaración del paquete, el autor y la declaración de la clase.

- La sección morada es la que tiene el método main, y dentro de él se crea un objeto de la clase Interfaz *(*que se explicara en el siguiente ejemplo de código 4), una vez con el objeto creado se invoca al método apariencia, para que se construya la apariencia gráfica del programa.

Analiza el siguiente código, que tiene la sintaxis del ejemplo de la clase Inicio explicada en el párrafo anterior:

Ejemplo de código de clase inicio.

```
package Codigo;

/**
 * @author UnADM
 */

public class Inicio {

    public static void main(String[] args){
        Apariencia.Interfaz i=new Apariencia.Interfaz();
        i.apariencia();
    }

}
```

En la sección anterior se describió la clase Inicio del programa de ejemplo para el envío de consultas, ahora se seguirá con la descripción y presentación de la clase Interfaz, la cual crea la apariencia gráfica del programa.

Si se observa el código de cada clase que se presenta, se verá que cada clase ha sido construida en paquetes diferentes, para separar la funcionalidad de la conexión a base de datos y la apariencia gráfica, por ello al comunicarse con otra clase primero se coloca el nombre del paquete donde se encuentra.

Ahora se verá la descripción de la clase Interfaz, la cual, como ya se ha mencionado, crea la apariencia gráfica del programa de ejemplo para el envío de consultas, esta clase cuenta con las siguientes secciones de código:

- La sección marcada en azul es la declaración del paquete, la importación de las librerías necesarias, declaración de las clases y las llaves de cierre requeridas.
- La sección marcada en naranja es la declaración de los componentes gráficos que se utilizarán para el ejemplo, estos componentes están declarados a nivel de clase para tener la posibilidad de manipular los componentes desde cualquier parte de la clase.

- La sección amarilla contiene la declaración del marco donde se colocarán todos los componentes gráficos.
- La sección morada tiene la configuración de los componentes gráficos, la añadidura al panel, y la adición del panel al marco.
- La sección verde claro tiene la adición del manejo de eventos del botón de validar, el cual llama al evento *validar();* que se describirá más abajo.
- La sección rojo claro tiene la adición del manejo de eventos del botón de **almacenar**, el cual llama al evento *almacenar();*que se describirá más abajo.
- La sección verde es el método de validar, el cual toma los valores de las cajas de texto y las envía a la clase conexión.

Examina el siguiente código que pertenece a la clase que crea la apariencia gráfica del programa de ejemplo para conexión a base de datos. Esta clase contiene elementos que reconocerás con facilidad, ya que la construcción de apariencias gráficas y eventos fueron los temas tratados en las dos unidades anteriores.

Ejemplo de código Clase Interfaz

```
package Apariencia;

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

/**
 * @author UnADM
 */

public class Interfaz {

    JFrame frame=new JFrame("Login");
    JPanel pnl=new JPanel();
    JLabel lbl1=new JLabel();
    JLabel lbl2=new JLabel();
    JTextField txtUs=new JTextField();
```

```
    JPasswordField txtC=new JPasswordField();
    JButton btnV= new javax.swing.JButton();
    JButton btnA= new javax.swing.JButton();

    public void apariencia(){

        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
        frame.setSize(300,200);

        frame.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });


        lbl1.setText("Usuario");
        lbl2.setText("Contraseña");
        btnV.setText("Validar");
        btnA.setText("Almacenar");

        GridLayout acomodo = new GridLayout(3,2);
        pnl.setLayout(acomodo);

        pnl.add(lbl1);
        pnl.add(txtUs);
        pnl.add(lbl2);
        pnl.add(txtC);
        pnl.add(btnV);
        pnl.add(btnA);

        frame.add(pnl, BorderLayout.CENTER);

        btnV.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e){
                 validar();
            }
        });

        btnA.addMouseListener(new MouseAdapter() {
```

```
            @Override
            public void mouseClicked(MouseEvent e){
                 almacenar();
            }
        });

    }

    public void validar(){
        String usuario = txtUs.getText();
        String contrasena=txtC.getText();

        Datos.Conexion.con(usuario, contrasena);
    }

    public void almacenar(){
        String usuario = txtUs.getText();
        String contrasena=txtC.getText();

        Datos.Conexion.inserta(usuario, contrasena);
    }

}
```

Hasta este momento se han revisado las primeras dos clases del programa de ejemplo para el envío de consultas, la primera en describirse fue la clase Inicio, la cual crea el método **main** que inicializa el programa e invoca la construcción de la interfaz gráfica de usuario; la siguiente clase que se explicó fue la clase Interfaz, ahora se explicará la tercer clase del ejemplo, que se llama Conexión; en ella se tiene la creación de la conexión del programa con la base de datos mediante la cadena de conexión, se tiene también el envío de sentencias y la recepción de respuesta de la base de datos hacia el programa.

La clase Conexión consta de:

- La sección marcada en azul es la declaración del paquete, la importación de las clases necesarias, declaración de la clase y las llaves de cierre requeridas.
- La sección morada es el método inserta, que recibe como parámetro dos cadenas (*us, con*), también se marcó el control de excepciones en este bloque (*try, catch*). El manejo de excepciones debe utilizarse cuando se tiene código sensible a errores de ejecución, en el caso de conexión a bases de datos, siempre se utiliza el manejo de excepciones porque puede ocurrir que la conexión no se realice, o que la sentencia enviada tenga un error, o los datos introducidos por el usuario no correspondan a los esperados por la base de datos.

- La sección naranja tiene la conexión a la base de datos.
- La sección verde crea el Statement y construye la sentencia SQL que será enviada a la base de datos. Y posteriormente la envía a ejecutar, nota que se ejecuta mediante la sentencia *executeUpdate*, ya que ésta es la que puede enviar un INSERT, UPDATE o DELETE, o una sentencia SQL que no devuelven ningún resultado.
- La sección roja informa que se llevó a cabo la ejecución de la inserción mediante una caja de diálogo.
- En amarillo se tiene el cierre del Statement y de la conexión, es importante que después de ser utilizados se cierren, para cortar la comunicación de información entre el programa y la base de datos y evitar posibles filtros de información innecesaria.

Ejemplo de código clase conexión

```
package Datos;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.swing.JOptionPane;

/**
 * @author UnADM
 */

public class Conexion {

    public static void inserta(String us, String con){

        try{
            Class.forName("org.gjt.mm.mysql.Driver");
            Connection conexion =
DriverManager.getConnection("jdbc:mysql://localhost/ejemplo",
"root", "root");
            Statement st = conexion.createStatement();
            String insercion="INSERT INTO ejemplo.usuario VALUES
('"+us+"','"+con+"');";

            st.executeUpdate(insercion);
            JOptionPane.showMessageDialog(null, "Insercion
correcta", "OK", JOptionPane.INFORMATION_MESSAGE);
            st.close();
```

```
            conexion.close();
        }
        catch (Exception e){
            JOptionPane.showMessageDialog(null, "Error de
insercion", "Error", JOptionPane.ERROR);
      }

    }

}
```

Como se pudo observar, la creación de programas que realizan consultas a bases de datos requiere de la conjunción de todos los temas tratados a lo largo de la unidad, por ello es muy importante que si tienes dudas vuelvas a revisar el ejemplo, recuerda que los nombres de la base de datos y de los parámetros para crear la cadena de conexión son de ejemplo; si tú al crear la base de datos cambiaste algunos nombres también deberás actualizarlos en el código presentado y crea la cadena de conexión en base a los datos correspondientes a tu propio gestor de base de datos.

3.2.3. Presentación de resultados

En la presentación de resultados se deben aplicar todos los conocimientos adquiridos durante la unidad, ya que como su nombre lo dice se debe realizar la presentación de los resultados que proporciona la base de datos como respuesta a la consulta realizada, lo cual es importante para que un programa obtenga datos desde la BD, es decir se extraiga información y la muestre en el programa.

Lo primero que se debe realizar para extraer resultados desde una base de datos es crear un objeto de tipo ResultSet (este tipo de objetos fue creado por Java exclusivamente para almacenar los resultados proporcionados por la base de datos después de una consulta). Éste es un objeto que como una tabla dinámica de datos crece y se ajusta a los datos que retorne la base de datos, después de ejecutar una sentencia SQL. Por ejemplo si se envía a la base de datos una sentencia SQL solicitando toda la información de una tabla (*Selecta all FROM tabla.bd*), no importa si en ese momento la tabla tiene sólo 10 registros o 10000, el ResultSet se ajustará a lo que la base de datos regrese.
Una vez que la base de datos retorna los datos de respuesta a la consulta, éstos se almacenan en el ResultSet, lo que debes hacer para revisar los datos de resultado es avanzar, ya que por default Java se posiciona antes de la primera fila del ResultSet. Para avanzar en el objeto y recorrer los resultados Java cuenta con el método next que mueve el cursor (o posición actual en el ResultSet) a la fila siguiente, y devuelve un false cuando no hay más filas en él, de esta manera cuando la posición en el ResultSet regresa false,

significa que has terminado de recorrer los resultados. Es recomendable utilizar un ciclo while para recorrer todos los resultados (ORACLE, 2012).

A continuación se muestra un ejemplo para la visualización de resultados donde veras que para obtener resultados, primero se debe crear la conexión y enviar la sentencia mediante el Statement. Después se declara el ResultSet en nulo, para que inicialmente no contenga nada. Después se iguala el ResultSet a lo que retorna la ejecución de la consulta para hacer que el ResultSet contenga la información que la base de datos ha retornado.

En ese momento el ResultSet contiene, dentro de sí, lo que la base de datos haya dado como respuesta a la sentencia enviada.

Revisa detalladamente el ejemplo de uso del ResultSet, analiza las secciones y su descripción, para que puedas comprender su sintaxis y uso.

Ejemplo de código ResultSet

```
Class.forName("org.gjt.mm.mysql.Driver");
            Connection conexion =
DriverManager.getConnection("jdbc:mysql://localhost/ejemplo",
"root", "root");
            Statement st = conexion.createStatement();
            String consulta="SELECT * FROM ejemplo.usuario WHERE
idusuario='"+us+"' && contra='"+con+"';";

            ResultSet rs=null;
            rs = st.executeQuery( consulta );
```

Para comprender mejor el proceso completo de la obtención de resultados (el cual implica crear conexión, se declara la cadena de conexión, enviar consultas y obtener respuesta de la BD) en el siguiente ejemplo para completarlo debes utilizar las clases del ejemplo de código anterior (3, 4 y 5), y sólo agregar la clase conexión anterior (5) para que funcione correctamente. A continuación se presenta la explicación del ejemplo de código 6 para la obtención de resultados:

- La sección marcada en azul es la declaración del paquete, la importación de las librerías necesarias, declaración de la clase y las llaves de cierre requeridas.
- La sección morada tiene la declaración del método con, y el control de excepciones requerido para el manejo de bases de datos.
- La sección naranja crea la conexión a la base de datos.
- La sección verde crea el Statement, así como la definición de la consulta SQL que se enviará a la base de datos. Esta consulta retornara los datos del registro, si el usuario y la contraseña ingresados por el usuario son iguales a algún registro de la base de datos.

- La sección amarilla contiene la declaración del ResultSet, su igualación a lo que retorne la sentencia *(rs = st.executeQuery(consulta);)* y una comparación para saber si el ResultSet ya no es nulo como en su declaración *(rs != null)* y si contiene algo *(rs.next())* en cuyo caso significa que si se encontró un registro con un usuario y contraseña idénticos a los ingresados, en ese caso se enviará un mensaje de conformación al usuario mediante una caja de diálogo y de lo contrario también se le informará al usuario de su error.

Ejemplo de código obtención de resultados

```
package Datos;


import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.swing.JOptionPane;

/**
 * @author UnADM
 */

public class Conexion {


    public static void con(String us, String con){
        try{
            Class.forName("org.gjt.mm.mysql.Driver");
            Connection conexion =
DriverManager.getConnection("jdbc:mysql://localhost/ejemplo",
"root", "root");
            Statement st = conexion.createStatement();
            String consulta="SELECT * FROM ejemplo.usuario WHERE
idusuario='"+us+"' && contra='"+con+"';";

            ResultSet rs=null;
            rs = st.executeQuery( consulta );

            if (rs != null && rs.next()){
               JOptionPane.showMessageDialog(null, "Validacion
correcta!!!", "OK", JOptionPane.INFORMATION_MESSAGE);
            }
```

```
            else{
               JOptionPane.showMessageDialog(null, "Error de
validacion ", "Error", JOptionPane.ERROR_MESSAGE);
            }

            rs.close();
            st.close();
            conexion.close();
        }
        catch (Exception e){
            JOptionPane.showMessageDialog(null, "Validacion
incorrecta", "Error", JOptionPane.ERROR);
        }

    }
```

Ahora se verá un ejemplo en el que se obtienen varios datos como respuesta de la consulta SQL y se muestran en una tabla gráfica. Este ejemplo cuenta con dos clases una llamada Mostrar que es la que invoca la apariencia y funcionalidad del programa situado en otra clase llamada guiMostrar.

Pero primero revisa el ejemplo de código de la clase Mostrar:

- La sección marcada en azul es la declaración del paquete, la importación de las clases necesarias, declaración de la clase y las llaves de cierre requeridas.
- La sección morada es la que tiene el método main, y dentro de éste se crea un objeto de la clase guiMostrar, una vez con el objeto creado se invoca al método apariencia, para que se construya la apariencia gráfica del programa.

Nota: En este ejemplo al igual que en los anteriores (ejemplo de código 3, 4, 5 y 6) se tienen las clases divididas en paquetes diferentes.

Ejemplo de código Clase Mostrar

```
package Codigo;

/**
 * @author UnADM
 */
public class Mostrar {

    public static void main(String[] args){
        Apariencia.guiMostrar m=new Apariencia.guiMostrar();
        m.apariencia();
```

```
        }

}
```

Siguiendo con la descripción del ejemplo donde se obtienen varios datos como respuesta, como ya se ha mencionado, este ejemplo cuenta con dos clases: Mostrar y guiMostrar. Esta última es la clase que contiene la apariencia gráfica del programa, su funcionalidad; se describirá a continuación:

- La sección marcada en azul es la declaración del paquete, la importación de las clases necesarias, declaración de la clase y las llaves de cierre requeridas.
- La sección morada es la que construye parte de la apariencia, se tiene la creación y configuración del *marco*, y del botón, el cual en su manejo de eventos llama al método mostrarRegistros();
- En rojo está marcada la creación de la tabla que mostrará los datos obtenidos de la consulta. Para crear una tabla se debe construir su modelo, el cual es la estructura y datos que tendrá la tabla, y también se debe tener un scrollPane, el cual es un panel que hace que si la tabla generada tiene muchos registros y no se alcanza a mostrar en el panel creado, este panel habilitará barras de desplazamiento, para poder moverse a lo largo y ancho de la tabla. Al analizar el código, el modelo se declara a nivel de clase, porque se requiere modificarlo en los dos métodos, de esta manera se tiene acceso a él sin problema, el método addColumn del modelo es para colocar el nombre de las columnas que tendrá la tabla (cabeceras). Los datos que contendrá la tabla se añaden con el método addRow, el cual agrega una fila completa a la tabla, si se observa esa fila que se añade es un objeto de datos que contiene lo que el ResultSet trae, ya con el nombre de las columnas y las filas agregadas el modelo se agrega a la tabla, y la tabla se coloca dentro del scrollPane y es este último el que se agrega al marco.
- El código marcado en verde es el método mostrarRegistros(), el cual cuenta con un manejo de excepciones para los posibles errores con la conexión y manipulación de la base de datos. La conexión se marcó más clara, y el Statement más obscuro. En amarillo se tiene la declaración del ResultSet, así como la ejecución de la consulta (*st.executeQuery(consulta);*) colocando los datos en el *ResultSet (rs =).* Una vez con los datos en el ResultSet se van a "vaciar" los datos a la tabla, para hacerlo se utiliza un ciclo while, el cual se repetirá mientras el ResultSet tenga una siguiente posición *(rs.next())*, es decir, que mientras exista otro registro el ciclo continua, y cuando ya no haya ninguno se parará. Para sacar los datos se obtienen del ResultSet por medio de los índices de la columna en la que se encuentran, recuerda que el ResultSet es una tabla dinámica entonces cada registro es una fila, y el contenido de cada campo está acomodado por columnas, en este caso se tiene dos columnas, por lo que se tienen los índices 1 y 2 (*rs.getString(1),rs.getString(2)*).

- Por último también en verde se tiene el cierre del ResultSet, el Statement y la conexión.

A continuación, se muestra el código que se explicó previamente, en el que se debe revisar cada una de las porciones de código marcadas en colores y compararlo (el código) con la descripción correspondientes, para una mejor comprensión.

Ejemplo de código clase guiMostrar

```
package Apariencia;

import java.awt.BorderLayout;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTable;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.swing.JScrollPane;
import javax.swing.table.DefaultTableModel;

/**
 * @author UnADM
 */
public class guiMostrar {

    DefaultTableModel modelo= new DefaultTableModel();

    public void apariencia(){
        JFrame frame=new JFrame("Datos sacados desde BD");
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
        frame.setSize(300,300);
        frame.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
```

```
        JButton boton=new JButton();
        boton.setText("Ver registros");
        boton.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e){
                 mostrarRegistros();
            }
        });

        modelo.addColumn("Usuario");
        modelo.addColumn("Contraseña");
        JScrollPane scroll=new JScrollPane();
        JTable tabla = new JTable();
        tabla.setModel(modelo);
        scroll.setViewportView(tabla);
        frame.add(boton, BorderLayout.SOUTH);
        frame.add(scroll, BorderLayout.CENTER);
    }

    public void mostrarRegistros(){
        try{
            Class.forName("org.gjt.mm.mysql.Driver");
            Connection conexion =
DriverManager.getConnection("jdbc:mysql://localhost/ejemplo",
"root", "root");
            Statement st = conexion.createStatement();
            String consulta="SELECT * FROM ejemplo.usuario;";

            ResultSet rs=null;
            rs = st.executeQuery( consulta );

            while(rs.next()){
                Object[] data = {rs.getString(1),rs.getString(2)
};
                modelo.addRow(data);
            }
            rs.close();
            st.close();
            conexion.close();
        }
        catch (Exception e){
            System.out.println("error "+e);
```

```
            }
        }

}
```

Al ejecutar el programa previamente explicado la salida que se mostrará será algo similar a lo que está en la siguiente imagen. Los datos mostrados van a variar dependiendo de los registros de usuario y contraseña que hayas ingresado al ejecutar los ejemplos presentados.

Salida de la ejecución de la clase Mostrar

Por último no queda más que recordar que para la representación de datos es importante el uso de tablas, ya que en ellas podrás desplegar la información obtenida como resultado de la ejecución de sentencias SQL. De modo tal que ya podrás realizar diferentes consultas para así obtener y mostrar información en los programas que desarrolles que estén conectados a bases de datos.

Cierre de la Unidad

Has concluido la tercera y última unidad de la asignatura Programación orientada a objetos II; en esta unidad llamada Base de datos observaste como conectar un programa a una base de datos y con ello insertar, eliminar, modificar y mostrar información.

Como parte de esta unidad primero realizaste un repaso de las sentencias SQL, diferenciando entre aquellas sentencias que crean bases de datos (DDL, lenguaje de definición de datos) y las sentencias que manipulan (DML, lenguaje de manipulación de datos) la información ya contenida en las bases de datos.

Después del repaso general sobre sentencias SQL, aprendiste como crear una conexión entre un programa y una base de datos, utilizando la cadena de conexión, el cual requiere la ubicación, el nombre de usuario y contraseña de la base de datos para poder acceder a ésta. También te percataste que una vez con la conexión creada se pueden enviar consultas a la base de datos.

El último tema tratado fue el de presentación de resultado en el que además de crear la conexión y enviar consultas a la base de datos se ha mostrado como extraer información de ella y desplegarla de manera visual (mediante tablas) al usuario.

Como se revisó el último programa de ejemplo, se utilizaron todos los temas que se han tratado a lo largo de la materia de programación. El conocimiento adquirido se emplea de manera escalonada, es decir, cada nuevo tema incluye los anteriores, por lo que si tienes dudas respecto a alguno de los temas presentados, es recomendable se repasen, con la finalidad de prepararte para la siguiente asignatura Programación orientada a objetos III.

Para saber más

Es importante que instales un IDE en tu computadora personal para que pases todos los ejemplos de código y veas cómo funcionan, de esta manera podrás analizar el funcionamiento de los códigos presentados.

Se recomienda que instales NetBeans 7.0, como IDE, por su facilidad de uso:

- Oracle, NetBeans IDE 8.0.2 Download. Recuperado de https://netbeans.org/downloads/

Para llevar a cabo las prácticas presentadas además del IDE requieres instalar un manejador de base de datos, se recomienda instalar MySql (MySQL Community Server), y para mayor facilidad de uso se recomienda instalar también una herramienta gráfica para su control (MySQL Workbench), ambos serán encontrados en la siguiente dirección electronica:

- Oracle, MySQL Downloads. Recuperado de: https://www.mysql.com/downloads/

Si quieres estudiar más a fondo sobre los temas de bases de datos, puedes consultar el material de tus asignaturas de base de datos, y también puedes buscar información en

páginas de apoyo, se sugiere que utilices la información encontrada en la siguiente dirección electrónica:

- w3schools, SQL Tutorial. Recuperado de http://www.w3schools.com/sql/default.asp

Por último, no queda más que recordar que los métodos de cada uno de los objetos y clases vistas en esta asignatura son sólo parte del total de existentes, si requieres solucionar problemas que ocupan otras acciones puedes revisar el tutorial brindado por los dueños de Java, para que ahondes más en todas las bondades que el lenguaje te ofrece. Este tutorial lo podrás encontrar en la siguiente dirección electrónica:

- Oracle, The Java Tutorials. Recuperado de http://docs.oracle.com/javase/tutorial/

Fuentes de consulta

- Ceballos, F. (2006). *Interfaces gráficas y aplicaciones para internet* (2ª ed.). España: RA-MA.
- De Miguel, A. y Piattini, M. (1999). *Fundamentos y modelos de base de datos* (2ª ed.). España: RA-MA.
- Oracle (2011). *The JAVA Tutorials*. Estados Unidos de América: Oracle.