UDE
UNIVERSIDAD DE LA EMPRESA

Bases de Datos 3

- Docentes: Federico Gómez, Diego Siri
- Carreras: Licenciatura & Ingeniería en Informática
- Año de la carrera: 3º

UDE
UNIVERSIDAD DE LA EMPRESA

Capítulo 3:

Acceso a BD en Arquitecturas de 1 Capa

UDE
UNIVERSIDAD DE LA EMPRESA

Introducción:

En este capítulo estudiaremos los fundamentos para el desarrollo de aplicaciones que utilizan ***bases de datos*** como mecanismo de ***persistencia*** en ***arquitecturas lógicas de 1 capa***.

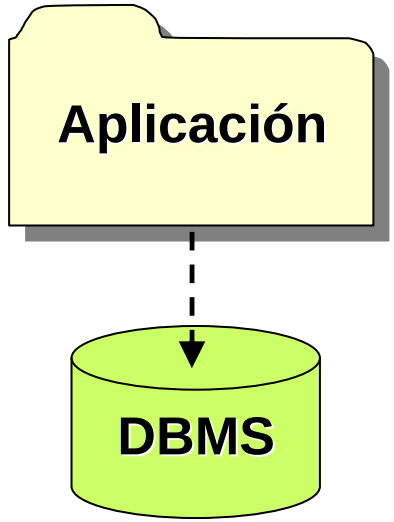
De acuerdo a lo visto en el capítulo 1, los sistemas desarrollados siguiendo arquitecturas lógicas de 1 capa son tales que acceden al mecanismo de persistencia (bases de datos) directamente desde la interfaz de usuario.

Recordar que no existe relación específica entre la ***arquitectura lógica*** y la ***arquitectura física*** de una aplicación. Pueden existir aplicaciones de 1 capa tanto ***standalone*** como ***distribuidas***. Lo que las distingue es el hecho de que fusionan presentación, lógica y persistencia, independientemente de la cantidad de equipos.

Características de las aplicaciones en 1 capa:

Las aplicaciones desarrolladas según una arquitectura lógica de 1 capa poseen las siguientes características:

- El acceso a la persistencia se realiza directamente desde la interfaz de usuario.
- La lógica de los requerimientos es resuelta en forma conjunta a la interacción con el usuario y el acceso a la persistencia.
- A nivel de código, no existe un límite claro entre presentación de datos, lógica de requerimientos y acceso a la persistencia.



Cuando la persistencia es manejada usando ***bases de datos***, se dice habitualmente que estas aplicaciones fusionan el ***front-end*** (interfaz de usuario) con el ***back-end*** (base de datos).

UDE
UNIVERSIDAD DE LA EMPRESA

Ventajas de las aplicaciones en 1 capa:

- Útiles en ***prototipos*** de aplicaciones, donde interesa mostrar al cliente parte de la interfaz de usuario con algo de funcionalidad.
- Proyectos ***caseros*** o de ***pequeño porte***, en los que se busca un desarrollo rápido y el costo de separar en capas puede ser mayor al beneficio.

Fueron muy populares en los inicios, cuando predominaban las arquitecturas ***standalone*** y las aplicaciones eran de un tamaño pequeño. No había necesidad de separar en capas.

También se utilizaron en las primeras arquitecturas ***distribuidas***, generalmente ***Cliente-Servidor***, donde había un servidor central que albergaba la BD y "clientes tontos" que lo accedían, ejecutaban consultas y listaban al usuario los resultados.

UNIVERSIDAD DE LA EMPRESA

Desventajas de las aplicaciones en 1 capa:

- Muy baja ***reusabilidad***. Al no existir componentes claramente diferenciados, es muy difícil reutilizarlos para otras aplicaciones.
- Muy baja ***mantenibilidad***. Al no existir capas, cualquier cambio (ya sea de interfaz, lógica o persistencia), por mínimo que sea, puede implicar **modificar** mucho código fuente.
- Muy baja ***portabilidad***. Al no existir capas, migrar la aplicación a una nueva interfaz de usuario (por ejemplo: ventanas escritorio a páginas web) o a una nueva persistencia (por ejemplo: cambiar de DBMS) puede implicar **re-escribir** mucho código fuente.

Conforme las aplicaciones han ido creciendo en complejidad y en tamaño, las arquitecturas de 1 capa han ido perdiendo utilidad, siendo paulatinamente suplantadas por arquitecturas de 2 y 3 capas (a estudiar más adelante en el curso).

UDE
UNIVERSIDAD DE LA EMPRESA

Ejemplos de aplicaciones en 1 capa:

Veremos dos ejemplos de aplicaciones en 1 capa desarrolladas en **Java** que usan ***bases de datos*** como mecanismo de persistencia, interactuando a través de **JDBC**.

Primer ejemplo:

Se trata de una aplicación ***standalone*** de escritorio que registra las horas de entrada y de salida de los empleados de un comercio.

La aplicación le muestra al empleado una ventana donde ingresar su cédula, y registra en una tabla de la base de datos la pareja **(cédula, hora)**.

En caso de que la cédula no pertenezca a un empleado registrado, se emitirá un mensaje de error. La aplicación **no** realizará otros chequeos, tales como verificar el total de marcas que el empleado lleva realizadas en el día o la cantidad de horas trabajadas.

Ejemplos de aplicaciones en 1 capa (continuación):


```
public void actionPerformed (...)
{
    abrir conexión con BD
    verificar cédula del empleado
    Si es correcta entonces
        obtener hora actual
        registrar (cédula, hora)
        mensaje de éxito
    Sino
        mensaje de error
    cerrar conexión con BD
}
```

Tabla Empleados

Cédula	Nombre
1.222.333	Juan
2.333.444	Teresa
4.555.666	Laura

Tabla Registro

Cédula	Hora
1.222.333	10:00
4.555.666	13:40
1.222.333	18:00

UDE
UNIVERSIDAD DE LA EMPRESA

Ejemplos de aplicaciones en 1 capa (continuación):

Acceso a la BD desde el método **actionPerformed**:

```
url = "jdbc:mysql://localhost:3306/Comercio";
user = "root"; password = "root";

Connection con = DriverManager.getConnection
                              (url, user, password);
String query = "SELECT cedula FROM Empleados "
             + "WHERE cedula = ?";
PreparedStatement pstmt1 =
         con.prepareStatement (query);
pstmt1.setString (1, cedula);
ResultSet rs = pstmt1.executeQuery ();

if (rs.next ())
   cedulaCorrecta = true;
```

UDE
UNIVERSIDAD DE LA EMPRESA

Ejemplos de aplicaciones en 1 capa (continuación):

Acceso a la BD desde el método **actionPerformed** (continuación):

```
rs.close();
pstmt1.close();
if (cedulaCorrecta) {
    String insert = "INSERT INTO Registro " +
                    "(cedula, hora) VALUES (?,?)";
    PreparedStatement pstmt2 =
              con.prepareStatement(insert);
    pstmt2.setString(1, cedula);
    pstmt2.setTime(2, hora);
    pstmt2.executeUpdate();
    pstmt2.close();
}
con.close();
```

UDE
UNIVERSIDAD DE LA EMPRESA

Mejorando cualidades del SW en aplicaciones en 1 capa:

El ejemplo anterior adolece de algunos problemas que afectan negativamente la ***calidad*** del código:

- **Todo** el código fuente está escrito dentro del método encargado de procesar el evento.
- Hay datos que están ***hard-code*** (url, usuario y password de la base de datos).
- El código encargado de armar los textos de las consultas y de ejecutarlas está en la **misma** clase de la ventana.
- ¿Otros inconvenientes?

Aún cuando la aplicación ***sigue estando*** en una ***arquitectura de 1 capa***, igualmente se pueden realizar algunas modificaciones con la finalidad de mejorar un poco la calidad del código.

UNIVERSIDAD DE LA EMPRESA

Mejorando cualidades del SW en aplicaciones en 1 capa (cont):

1) Cargar url, usuario y password de un ***archivo de configuración*** (archivo **`.properties`** de **Java**) al momento de crear la ventana y guardarlos como atributos de la misma, para así no tener que cargarlos en cada nueva conexión:

```
public VentanaRegistroHoras () {
    Properties p = new Properties ();
    String nomArch = "config/datos.properties";
    p.load (new FileInputStream (nomArch));
    url = p.getProperty ("url");
    user = p.getProperty ("user");
    password = p.getProperty ("password");
    ...
}
```

Mejorando cualidades del SW en aplicaciones en 1 capa (cont):

2) Crear una clase **Consultas** que defina el texto de cada consulta a realizar y no sobrecargar a la ventana con el armado del texto.

```
public class Consultas {
    public String verifyCedula() {
        String query = "SELECT cedula FROM " +
                "Empleados WHERE cedula = ?";
        return query;
    }
    public String insertRegistro() {
        String insert = "INSERT INTO Registro " +
                "(cedula, hora) VALUES (?,?)";
        return insert;
    }
}
```

Mejorando cualidades del SW en aplicaciones en 1 capa (cont):

3) Definir una clase **AccesoBD** que encapsule el acceso a la BD, y no sobrecargar a la ventana con la ejecución de las consultas.

```
public class AccesoBD {
    public boolean verifyCedula
       (Connection con, String cedula)
                    throws PersistenciaException
    {  /* verifico la cédula en la BD */  }

    public void insertRegistro
       (Connection con, String cedula, Time hora)
                    throws PersistenciaException
    {  /* inserto el registro en la BD */ }
}
```

UDE
UNIVERSIDAD DE LA EMPRESA

Mejorando cualidades del SW en aplicaciones en 1 capa (cont):

```
public boolean verifyCedula (...) throws ... {
    boolean cedulaCorrecta = false;
    try {
        Consultas consultas = new Consultas();
        String query = consultas.verifyCedula();
        PreparedStatement pstmt1 =
                con.prepareStatement(query);
        pstmt1.setString(1, cedula);
        ResultSet rs = pstmt1.executeQuery();
        if (rs.next()) cedulaCorrecta = true;
        rs.close();  pstmt1.close();
    } catch (SQLException e)
        throw new PersistenciaException(...);
    return cedulaCorrecta;
}
```

UNIVERSIDAD DE LA EMPRESA

Mejorando cualidades del SW en aplicaciones en 1 capa (cont):

```
public void insertRegistro (...) throws ... {
    try {
        Consultas consultas = new Consultas();
        String ins = consultas.insertRegistro();
        PreparedStatement pstmt2 =
                con.prepareStatement(ins);
        pstmt2.setString(1, cedula);
        pstmt2.setTime(2, hora);
        pstmt2.executeUpdate();
        pstmt2.close();
    } catch (SQLException e)
        throw new PersistenciaException(...);
}
```

UDE
UNIVERSIDAD DE LA EMPRESA

Mejorando cualidades del SW en aplicaciones en 1 capa (cont):

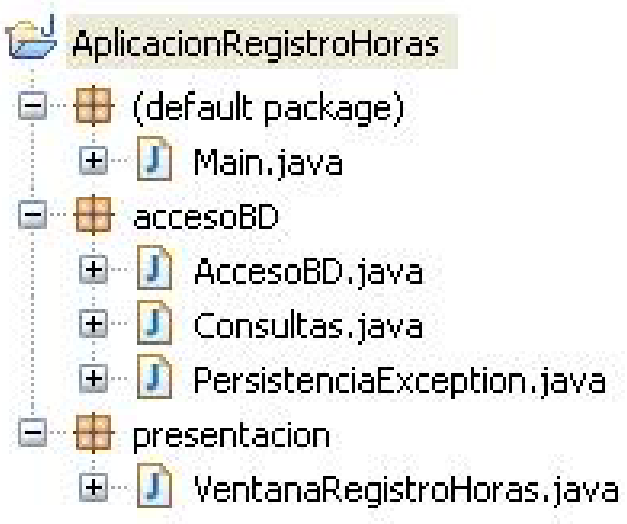
Acceso (mejorado) a la BD desde el método **actionPerformed**, luego de incorporadas las modificaciones anteriores:

```
Connection con = DriverManager.getConnection
                              (url, user, password);
AccesoBD accesobd = new AccesoBD ();
boolean cedulaCorrecta =
                accesobd.verifyCedula (con, cedula);
if (cedulaCorrecta)
   accesobd.insertRegistro (con, cedula, hora);
con.close ();
```

¿Qué otras mejoras se podrían hacer para seguir aumentando la calidad?

Mejorando cualidades del SW en aplicaciones en 1 capa (cont):

Una posible organización de las clases de la aplicación tras las modificaciones realizadas podría ser la siguiente:



- El ***package*** por defecto que contiene el programa principal **Main** encargado de arrancar la ejecución.
- El ***package*** **`accesoBD`** que contiene las clases vinculadas a la interacción con la base de datos (**AccesoBD**, **Consultas**, **PersistenciaException**).
- El ***package*** **`presentacion`** que contiene la clase encargada de interactuar con el usuario **VentanaRegistroHoras**.

UDE
UNIVERSIDAD DE LA EMPRESA

Mejorando cualidades del SW en aplicaciones en 1 capa (cont):

Los cambios realizados mejoraron la calidad del código fuente:

- Ya **no** está todo el código dentro del método **actionPerformed**.
- **No** hay más datos que estén ***hard-code***.
- El código encargado de armar los textos de las consultas ha sido delegado a la clase **Consultas**.
- El código encargado de ejecutar las consultas ha sido delegado a la clase **AccesoBD**.

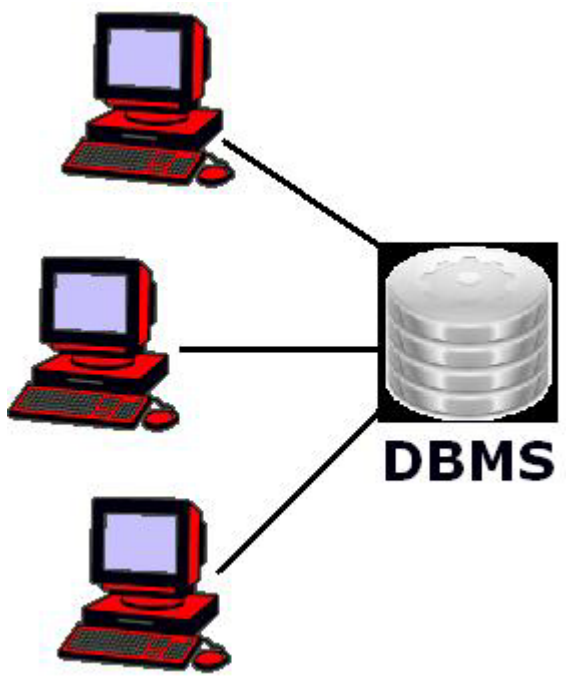
No obstante, la aplicación ***sigue estando en una arquitectura de 1 capa***. Si bien parte del código ha sido delegado a clases auxiliares, tanto la lógica del comportamiento como el acceso a la persistencia siguen siendo realizados desde la interfaz de usuario.

Ejemplos de aplicaciones en 1 capa (continuación):

Segundo ejemplo:

Se desea transformar la aplicación anterior en una aplicación ***distribuida*** de modo que ahora el **DBMS** resida en un equipo propio y existan ***varios*** equipos desde donde los empleados puedan registrar sus horas.

Se mantiene la ***arquitectura en 1 capa*** y el diseño de la versión anterior, sólo que ahora los equipos acceden al **DBMS** vía la red local en forma ***concurrente***.



¿Qué cambios es necesario realizar a la versión anterior para migrar a esta arquitectura distribuida?

UDE
UNIVERSIDAD DE LA EMPRESA

Ejemplos de aplicaciones en 1 capa (continuación):

Cambios necesarios para migrar a nueva arquitectura ***distribuida***:

1) A nivel del código fuente, el establecimiento de la conexión con la BD ***no cambia***. La **url** donde reside la base se carga de un ***archivo de configuración***. Basta con modificar dicha url en el archivo para establecer la conexión con el **DBMS** por la red.

Archivo **`datos.properties`** ***antes*** de la distribución:

```
url = jdbc:mysql://localhost:3306/Comercio
user = root
password = root
```

Archivo **`datos.properties`** ***luego*** de la distribución:

```
url = jdbc:mysql://nueva_url_DBMS:3306/Comercio
user = root
password = root
```

UDE
UNIVERSIDAD DE LA EMPRESA

Ejemplos de aplicaciones en 1 capa (continuación):

Cambios necesarios para migrar a nueva arquitectura ***distribuida***:

2) Dado que ahora hay varios ***clientes*** accediendo a la base en forma ***concurrente***, se hará necesario incorporar manejo de ***transacciones***, según lo visto en el capítulo 2.

Código de la primera versión **más** manejo de transacciones:

```
try {
  Connection con = DriverManager.getConnection
                              (url, user, password);
  con.setTransactionIsolation
          (Connection.TRANSACTION_SERIALIZABLE);
  con.setAutoCommit (false);
  AccesoBD accesobd = new AccesoBD ();
  ...
```

Ejemplos de aplicaciones en 1 capa (continuación):

Cambios necesarios para migrar a nueva arquitectura ***distribuida***:

```
    boolean cedulaCorrecta =
                accesobd.verifyCedula(con, cedula);
    if (cedulaCorrecta)
       accesobd.insertRegistro(con, cedula, hora);
    con.commit();
  }
  catch(Exception e) {
    con.rollback();
  }
  finally {
    con.close();
  }
```

UNIVERSIDAD DE LA EMPRESA

Ejemplos de aplicaciones en 1 capa (continuación):

- Cuando el cliente se conecta con la base de datos, solicita una ***transacción***, a efectos de cumplir con las propiedades ***A.C.I.D***. Finaliza haciendo ***commit*** o ***rollback***, según corresponda.
- El esquema propuesto funciona bien siempre y cuando sean ***pocos*** los clientes que se conecten con la BD. Para la realidad propuesta, es razonable suponer que esto será así.
- Las conexiones a la base son un ***recurso caro***. Si la cantidad de clientes aumenta considerablemente (ej: ***cientos*** de clientes), se necesitará un manejo más ***sofisticado*** de transaccionalidad.
- Tal manejo ***escapa*** al alcance de las ***arquitecturas en 1 capa*** y es más adecuado en arquitecturas en 2 y 3 capas. Recordar que las arquitecturas en 1 capa son apropiadas para ***prototipos*** y/o proyectos ***caseros***, de tamaño ***pequeño***.

UNIVERSIDAD DE LA EMPRESA

¿Lógica de requerimientos en la aplicación o en la BD?

Esta pregunta surge en todo desarrollo, ***independientemente*** de la ***arquitectura*** (tanto ***lógica*** como ***física***) de la aplicación. Existen dos enfoques posibles:

1. Lógica de los requerimientos resuelta principalmente a nivel del código de la ***aplicación***. La mayoría de los chequeos, controles, etc. se hacen en la aplicación y **no** en la BD.

2. Lógica de los requerimientos resuelta principalmente a nivel de la ***base de datos***. La mayoría de los chequeos, controles, etc. se realizan en la BD y **no** en la aplicación.

En principio, ningún enfoque es mejor que el otro. Cada uno tiene sus ventajas y sus desventajas. Usualmente los desarrolladores suelen preferir el primero, mientras que los administradores de DBMS suelen preferir el segundo.

¿Lógica de requerimientos en la aplicación o en la BD? (cont.):

Primer enfoque - Comportamiento en la *aplicación*

Ventajas:

- El desarrollador tiene mayor control sobre la ejecución.
- Consultas más pequeñas, más simples y más puntuales, mayor independencia del DBMS.
- No se sobrecarga al DBMS con ejecuciones complejas.

Desventajas:

- Mayor complejidad en el código de la aplicación.
- La aplicación debe encargarse del manejo de concurrencia (transacciones).
- Pueden desaprovecharse las bondades del DBMS para optimización de consultas y eficiencia.

¿Lógica de requerimientos en la aplicación o en la BD? (cont.):

Segundo enfoque - Comportamiento en la *base de datos*

Ventajas:

- Código de la aplicación más simple, se delega al DBMS la realización de chequeos y ejecuciones complejas.
- Minimiza (o elimina) manejo de concurrencia en aplicación.
- Aprovecha bondades del DBMS en optimización / eficiencia.

Desventajas:

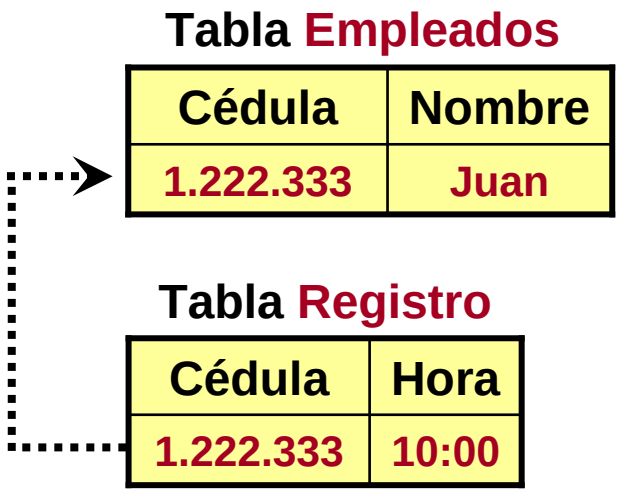
- El desarrollador tiene menor control sobre la ejecución.
- Las consultas son más complejas y se tiene una mayor dependencia del DBMS.
- Se delega al DBMS la ejecución de sentencias complejas.

¿Lógica de requerimientos en la aplicación o en la BD? (cont.):

Los dos ejemplos de ***aplicaciones en 1 capa*** propuestos siguieron el **primer enfoque** (comportamiento en la ***aplicación***). En ambos se controló la existencia del empleado a nivel de la aplicación.

Sin embargo, se podría haber exigido ***integridad referencial*** entre la tabla **Empleados** y la tabla **Registro**. Así, el DBMS produciría un error al intentar registrar la marca de un empleado que **no** existe.

```
CREATE TABLE Registro
( cedula VARCHAR(30) NOT NULL,
  hora TIME NOT NULL,
  PRIMARY KEY (cedula, hora)
  FOREIGN KEY (cedula)
    REFERENCES Empleados (cedula)
)
```



UDE
UNIVERSIDAD DE LA EMPRESA

¿Lógica de requerimientos en la aplicación o en la BD? (cont.):

Esto habría evitado la necesidad de tener la consulta que chequea la existencia del empleado:

```
SELECT cedula FROM Empleados WHERE cedula = ?
```

Bastaría solamente con tener la sentencia que inserta el registro:

```
INSERT INTO Registro (cedula, hora) VALUES (?,?)
```

Al intentar ejecutar esta sentencia para un empleado que no existe, el DBMS producirá un error, el cual será atrapado por la aplicación en forma de una **SQLException** en la clase **AccesoBD**.

P: ¿Cómo saber si la excepción fue por *integridad referencial* u otra razón (ejemplo: error de comunicación con BD)?

R: En la clase AccesoBD se debe consultar el *código de error* devuelto por el DBMS.

¿Lógica de requerimientos en la aplicación o en la BD? (cont.):

```
public void insertRegistro (...) throws ... {
    try {
        Consultas consultas = new Consultas();
        String ins = consultas.insertRegistro();
        .....
        pstmt2.executeUpdate();
        pstmt2.close();
    } catch (SQLException e) {
        if (e.getErrorCode() == 1452)
            error = "Empleado no existente";
        else
            error = "Error comunicación con BD";
        throw new PersistenciaException(error);
    }
}
```

UDE
UNIVERSIDAD DE LA EMPRESA

¿Lógica de requerimientos en la aplicación o en la BD? (cont.):

- Con este enfoque, la aplicación no necesita ejecutar la primer consulta, ***pero*** necesita identificar el código de error devuelto por el DBMS al ejecutar la sentencia de inserción (consultar la documentación del DBMS por una lista de códigos de error).
- En la segunda versión del ejemplo (con arquitectura ***distribuida***) ***deja de ser necesario*** el manejo de ***transaccionalidad***. Esto es porque ahora la aplicación ejecuta ***una única sentencia*** **SQL**.

Conclusión: Ambos enfoques presentan ventajas y desventajas. Según las características de la aplicación que se desee desarrollar se debe hacer un análisis previo del enfoque a adoptar.

A medida que avance el curso, veremos que el **primer enfoque** (comportamiento en la ***aplicación***) termina resultando preferible en términos de las ***cualidades del software***.