Departamento de Cs. e Ingeniería de la Computación

Universidad Nacional del Sur

Bases de Datos

Segundo Cuatrimestre de 2018

Proyecto Nº 2

Implementación de un sistema de base de datos bancario

Ejercicios

1. Implemente una base de datos en MySQL respetando el modelo relacional que acompaña este enunciado (ver apéndice A). El nombre de dicha base de datos deberá ser *"banco"* y los esquemas de las tablas deberán respetar los esquemas propuestos por dicho modelo relacional. Se deberán respetar los nombres de las relaciones y los atributos, así como las restricciones de llaves primarias y foráneas. **No se aceptará una base de datos que no respete estas convenciones (ver apéndice B.1**).

   Además deberá crear los siguientes usuarios que definen diferentes tipos de acceso al Servidor MySQL:

   - *admin*: Este usuario se utilizará para administrar la base de datos *"banco"* por lo tanto deberá tener acceso total sobre todas las tablas, con la opción de crear usuarios y otorgar privilegios sobre las mismas. Para no comprometer la seguridad se restringirá que el acceso de este usuario se realice sólo desde la máquina local donde se encuentra el servidor MySQL. El password de este usuario deberá ser *admin*.
   - *empleado*: Este usuario estará destinado a permitir el acceso de la aplicación de administración que utilizan los empleados del banco para administrar los clientes, préstamos, cajas de ahorro y plazos fijos. Para esto necesitará privilegios para:
     - sólo realizar consultas sobre: Empleado, Sucursal, Tasa_Plazo_Fijo y Tasa_Prestamo.
     - realizar consultas e ingresar datos sobre: Préstamo, Plazo_Fijo, Plazo_Cliente, Caja_Ahorro y Tarjeta.
     - realizar consultas, ingresar y modificar datos sobre: Cliente_CA, Cliente y Pago.

     Dado que el banco cuenta con varias sucursales distribuidas en diferentes ciudades, este usuario deberá poder conectarse desde cualquier dominio. El password de este usuario deberá ser *empleado*. **Importante:** Recuerde eliminar el usuario *vacío* (`drop user ''@localhost`) para poder conectarse con el usuario *empleado* desde localhost.
   - *atm*: Este usuario está destinado a permitir el acceso de los ATM, para que los clientes puedan consultar el estado de sus cajas de ahorro y realizar transacciones. Con el objetivo de ocultar la estructura de la base de datos, el usuario *atm* tendrá una visión restringida de la misma que solamente le permita ver información relacionada a las transacciones realizadas sobre las cajas de ahorro. A tal efecto, se deberá crear una *vista* con el nombre *trans_cajas_ahorro* que contenga la siguiente información:
     - Número y saldo de cada caja de ahorro.
     - Número, fecha, hora, tipo (*débito, extracción, transferencia, depósito*) y monto de cada transacción realizada sobre cada caja de ahorro. En caso que la transacción sea una transferencia la vista deberá contener el número de la caja de ahorro destino. Además, deberá contener el código de la caja donde fué realizada la transacción (salvo que sea un débito).

- Número de cliente, tipo y número de documento, nombre y apellido del cliente que realizó cada transacción (sólo para débito, extracción y transferencia).

El usuario *atm* tendrá privilegio de lectura sobre la vista *trans_cajas_ahorro*. Además este usuario deberá tener permiso de lectura y actualización sobre la tabla **tarjeta** (ver apéndice A) para poder controlar el ingreso de los clientes a los ATM y permitir que cambien el PIN de su tarjeta. Dado que los cajeros automáticos se encuentran distribuidos en diferentes ciudades, este usuario deberá poder conectarse desde cualquier dominio. El password de este usuario deberá ser *atm*.

Se **deberá entregar** un archivo de texto con el nombre *"banco.sql"* (en formato digital, no impreso) con la secuencia de sentencias para la creación de la base de datos, las tablas, la vista, los usuarios con nombre, password y privilegios correspondientes. Ademas deberá entregar un archivo de texto (en formato digital, no impreso) con el nombre *"datos.sql"* con una carga inicial de datos de prueba adecuados como para poder realizar consultas significativas sobre ellos y probar la vista y la aplicación (ejercicio 2).

2. Diseñe e implemente una aplicación en el lenguaje Java que se comunique con la base de datos *"banco"* y provea las siguientes funcionalidades:

   a) **Realizar consultas a la base de datos ingresando sentencias SQL.**
   Esta parte de la aplicación estará disponible sólo para el administrador del sistema por lo cual se requerirá el password correspondiente de forma oculta (password=*******, ver `JPasswordField`).

   En caso que el password sea incorrecto se deberá mostrar un mensaje de error adecuado. Si el password es el correcto, la interfaz de usuario deberá mostrar un área de texto (`JTextArea`) que permita introducir sentencias SQL arbitrarias (select, insert, delete, update, etc) mostrando el resultado (en el caso que corresponda) de la ejecución de dicha consulta en una tabla (`Jtable`). En caso que la consulta produzca un error, deberá mostrarse un mensaje explicando el error.

   Además deberá mostrar una lista (Por ejemplo `JList`) que contenga los nombres de todas las tablas presentes en la B.D. *"banco"*. Cuando se seleccione un ítem (nombre de una tabla) de esta lista, se mostrará en otra lista los nombres de todos los atributos de la tabla seleccionada (Ver sentencias `SHOW TABLES` y `DESCRIBE <nombre_tabla>` en el manual de MySQL).

   b) *Cajero automático (ATM): Consultas de saldo y movimientos sobre cajas de ahorro.*
   Esta parte de la aplicación simulará alguna de las funciones de un cajero automático. Deberá permitir que se ingrese un número de tarjeta y un número de PIN de forma oculta, y controlará que el número de PIN corresponda al número de tarjeta ingresado. En caso de que el PIN sea incorrecto deberá mostrarse un mensaje adecuado y permitir ingresar el PIN nuevamente. En caso que el PIN sea correcto se mostrará un menú que permitirá realizar las siguientes consultas:

   - *Consulta de Saldo*: mostrará el saldo actual de la caja de ahorro asociada a la tarjeta ingresada.
   - *Ultimos Movimientos*: mostrará un resumen en forma de tabla con las últimas 15 transacciones realizadas sobre la caja de ahorro. Dicho resumen deberá contener la siguiente información ordenada por fecha y hora en forma descendente:
     - Fecha y hora de la transacción.
     - Tipo de transacción.
     - Monto de la transacción: en caso que la transacción decremente el saldo (extracción, transferencia y débito), el monto deberá mostrarse como un número negativo (precedido por el signo "-").

- El código de caja donde fué realizada.
- Caja de ahorro destino, en caso de ser una transferencia.

- *Movimientos por período*: Permitirá ingresar dos fechas válidas y mostrará un resumen (con la misma información que se muestra en *Ultimos Movimientos*) con las transacciones realizadas entre las dos fechas ingresadas.
  Nota: El formato de la fecha y hora recuperado desde JAVA a través de JDBC puede variar según el formato de fecha que esté configurado en Windows. **La aplicación entregada deberá funcionar correctamente independientemente de la configuración de fecha y hora de Windows** (ver apéndice B.2)

c) *Administración de Préstamos*
Esta parte de la aplicación proveerá alguna de las funciones de administración que realizan los empleados del Banco. Inicialmente deberá permitir que se ingrese el número de legajo de un empleado y un password de forma oculta, y controlará que el password corresponda al número de legajo ingresado. En caso de que el password sea incorrecto deberá mostrarse un mensaje adecuado y permitir ingresar el password nuevamente. En caso que el password sea correcto (el empleado se considera *"logueado"*) se mostrará un menú que permitirá realizar las siguientes tareas:

- *Creación de Préstamos*: la interfaz permitirá seleccionar un cliente mediante su tipo y número de documento y controlará que el mismo no esté pagando un préstamo actualmente. En caso que el cliente tenga un préstamo vigente se deberá mostrar un mensaje y no se podrá crear un nuevo préstamo. Si el cliente no tiene préstamos vigentes se permitirá ingresar un monto que no supere el máximo monto superior de la tabla **Tasa_prestamo** (ver apéndice A) y se permitirá seleccionar una cantidad de meses que se corresponda con un período de los existentes en la tabla **Tasa_prestamo**. En base a esta información se podrá obtener la tasa de interes correspondiente y calcular el valor de la cuota (utilizando la fórmula mencionada en el proyecto 1). Con toda esta información se creará un nuevo préstamo con fecha actual, asociado al cliente ingresado y al empleado que se encuentra *"logueado"* actualmente.
  Además deberán cargarse todas las cuotas (pagos) asociadas al préstamo. La primer pago tendrá fecha de vencimiento un mes después de la fecha de creación del prestamo y el resto de los pagos tendrá fecha de vencimiento un mes después de la cuota anterior. *Por ejemplo: para un préstamo a pagar en 6 meses creado el día 21/9/2015 las fechas de vencimiento serán las siguientes: la cuota 1 vence el 21/10/2015, la cuota 2 vence el 21/11/2015, ..., la cuota 6 vence el 21/3/2016*. Para calcular las fechas de vencimiento puede utilizar la función de MySQL `date_add('2015-9-21', interval 1 month)`.
- *Pago de Cuotas*: la interfaz permitirá selecionar un cliente mediante su tipo y número de documento. Una vez seleccionado el cliente la aplicación deberá mostrar el número, valor y la fecha de vencimiento de todas las cuotas impagas (sin fecha de pago) del préstamo asociado a dicho cliente. Se deberá permitir seleccionar una o más cuotas y registrar el pago de las mismas.
- *Listado de clientes morosos*: mostrará un listado en forma de tabla con información de los clientes que se han atrasado en al menos 2 cuotas de su préstamo, esto es, existen al menos dos cuotas impagas cuya fecha de vencimento es menor a la fecha actual. Dicho resumen deberá contener la siguiente información:
  - Número de cliente, tipo y número de documento, nombre y apellido del cliente.
  - Número de préstamo, monto, cantidad de meses y valor de la cuota del préstamo asociado al cliente.
  - Cantidad de cuotas atrasadas de cada préstamo.

  La tabla deberá ordenarse según la columna seleccionada por el usuario, en forma ascendente o descendente (ver método `setAutoCreateRowSorter` de la clase `Jtable`).

Fechas y condiciones de entrega

- **Fechas límite de entrega:**
  - **Ejercicio 1 - 14 de Septiembre de 2018**: a través del curso **Moodle** de la materia (`https://moodle.uns.edu.ar/moodle`) y utilizando la tarea correspondiente habilitada para tal fin, se deberán subir los archivos de texto *"banco.sql"* y *"datos.sql"* solicitados en el **ejercicio 1**
  - **Ejercicio 2 - 4 de Octubre de 2018**: a través del curso **Moodle** de la materia y utilizando la tarea correspondiente habilitada para tal fin, se deberá subir un archivo comprimido (zip o rar) que contenga: los archivos fuentes de la aplicación solicitada en el **ejercicio 2**, el archivo JAR ejecutable correspondiente y los archivos *"banco.sql"* y *"datos.sql"* solicitados en el ejercicio 1.
- **Comisiones:** Los proyectos deben realizarse en comisiones de *dos alumnos* cada una. Las comisiones deberán ser *las mismas* para todas las entregas.
- **Importante:** La entrega en *tiempo y forma* de este proyecto es *condición de cursado* de la materia.

A. Modelo E-R y Modelo Relacional

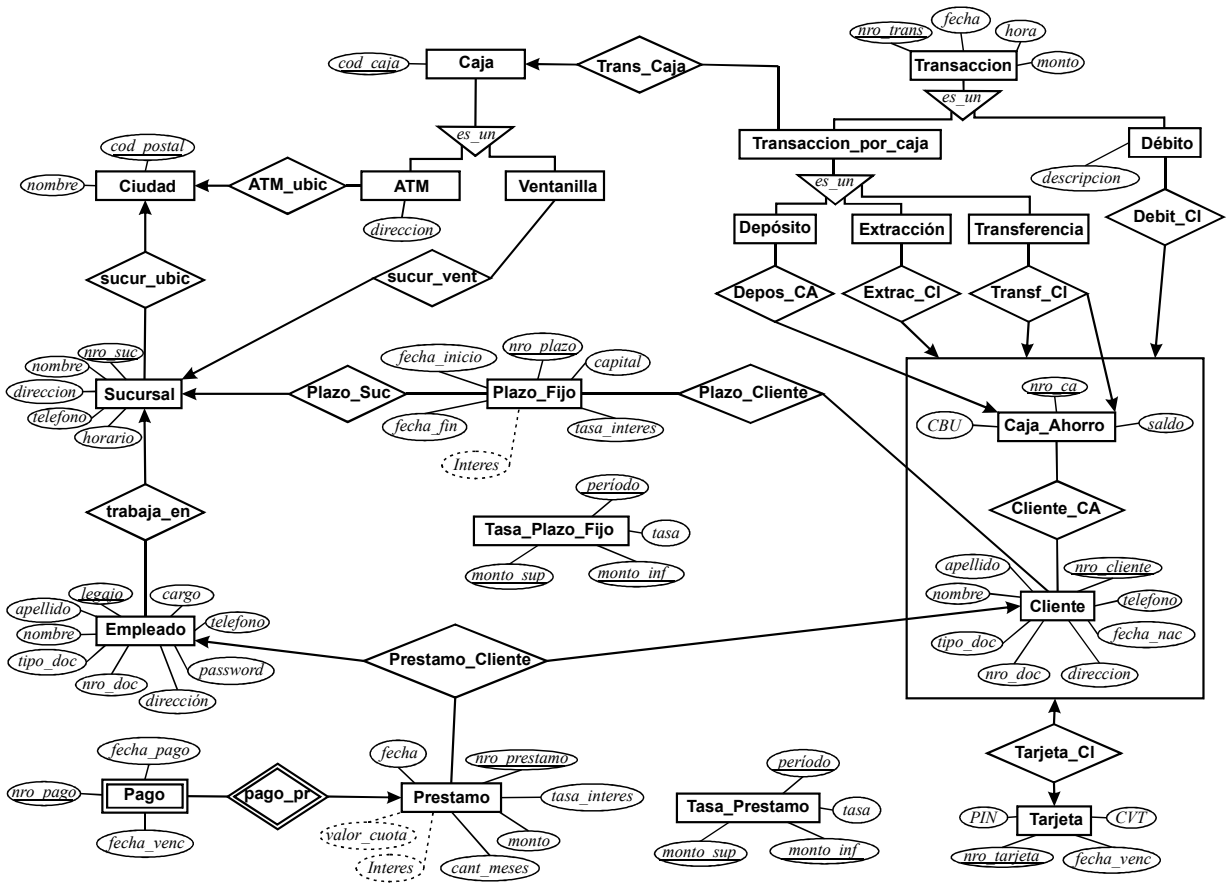

Figura 1: Modelo Entidad-Relación sistema bancario

- **Ciudad** (cod_postal, nombre)
  cod_postal es un natural de 4 cifras y nombre es una cadena de caracteres.

- **Sucursal** (nro_suc, nombre, direccion, telefono, horario, cod_postal)
  número es un natural de 3 cifras; nombre, direccion, telefono y horario son cadenas de caracteres; y cod_postal corresponde al código postal de una ciudad.

- **Empleado** (legajo, apellido, nombre, tipo_doc, nro_doc, direccion, telefono, cargo, password, nro_suc)
  legajo es un natural de 4 cifras; apellido, nombre, tipo_doc, direccion, telefono, cargo y password son cadenas de caracteres; nro_doc es un natural de 8 cifras; nro_suc corresponde al número de una sucursal. El campo password debe ser una cadena de 32 caracteres, para poder almacenarlo de forma segura utilizando la función de hash `MD5` *provista por MySQL(ver sección B.3)*

- **Cliente** (nro_cliente, apellido, nombre, tipo_doc, nro_doc, direccion, telefono, fecha_nac)
  nro_cliente es un natural de 5 cifras; apellido, nombre, tipo_doc, direccion y telefono son cadenas de caracteres; nro_doc es un numero de 8 cifras.

- **Plazo_Fijo** (nro_plazo, capital, fecha_inicio, fecha_fin, tasa_interes, interes, nro_suc)
  nro_plazo es un natural de 8 cifras; capital, tasa_interes e interes son reales positivos con 2 decimales; nro_suc corresponde a un numero de sucursal.

- **Tasa_Plazo_Fijo**(periodo, monto_inf, monto_sup, tasa)
  periodo es un natural de 3 cifras; monto_inf, monto_sup, tasa son reales positivos con 2 decimales.

- **Plazo_Cliente** (nro_plazo, nro_cliente)
  nro_plazo y nro_cliente corresponden a un número de plazo fijo y cliente respectivamente

- **Prestamo** (nro_prestamo, fecha, cant_meses, monto, tasa_interes, interes, valor_cuota, legajo, nro_cliente)
  nro_prestamo es un natural de 8 cifras; cant_meses es un natural de 2 cifras; monto, tasa_interes, interes y valor_cuota, son reales positivos con 2 decimales; legajo corresponde al legajo de un empleado y nro_cliente corresponde a un número de cliente.

- **Pago** (nro_prestamo, nro_pago, fecha_venc, fecha_pago)
  nro_prestamo corresponde a un número de préstamo; nro_pago es un natural de 2 cifras.

- **Tasa_Prestamo** (periodo, monto_inf, monto_sup, tasa)
  perido es un natural de 3 cifras; monto_inf, monto_sup y tasa son reales positivos con 2 decimales.

- **Caja_Ahorro** (nro_ca, CBU, saldo)
  nro_ca es un natural de 8 cifras, CBU es un natural de 18 cifras y saldo es un real positivo con 2 decimales.

- **Cliente_CA** (nro_cliente, nro_ca)
  nro_cliente corresponde a un número de cliente, nro_ca corresponde a un número de Caja de Ahorro.

- **Tarjeta** (nro_tarjeta, PIN, CVT, fecha_venc, nro_cliente, nro_ca)
  nro_tarjeta es un natural de 16 cifras; *PIN y CVT son cadenas de 32 caracteres, para poder almacenarlos de forma segura utilizando la función de hash* `MD5` *provista por MySQL(ver sección B.3); nro_cliente, nro_ca corresponden a un número de cliente y caja de ahorro presentes en la relación Cliente_CA.*

- **Caja** (cod_caja)
  cod_caja es un natural de 5 cifras.

- **Ventanilla** (cod_caja, nro_suc)
  cod_caja corresponde a una Caja y nro_suc corresponde a una sucursal

- **ATM** (cod_caja, cod_postal, direccion)
  cod_caja corresponde a una Caja, cod_postal corresponde a una ciudad y direccion es una cadena de caracteres.

- **Transaccion** (nro_trans, fecha, hora, monto)
  nro_trans es un natural de 10 cifras y monto es un real positivo con 2 decimales.

- **Debito** (nro_trans, descripcion, nro_cliente, nro_ca)
  nro_trans corresponde a un número de transacción; descripción es una cadena de caracteres; nro_cliente, nro_ca corresponde a un número de cliente y número de caja de ahorro presentes en la relacion Cliente_CA.

- **Transaccion_por_caja** (nro_trans, cod_caja)
  nro_trans corresponde a un número de transacción y cod_caja corresponde a una Caja.

- **Deposito** (nro_trans, nro_ca)
  nro_trans corresponde a un número de Transacción_por_caja; nro_ca corresponde a un número de Caja de Ahorro.

- **Extraccion** (nro_trans, nro_cliente, nro_ca)
  nro_trans corresponde a un número de transacción_por_caja; nro_cliente, nro_ca corresponde a un número de cliente y número de Caja de Ahorro presentes en la relacion Cliente_CA.

- **Transferencia** (nro_trans, nro_cliente, origen, destino)
  nro_trans corresponde a un número de transacción_por_caja; nro_cliente y origen corresponden a un número de cliente y número de Caja de Ahorro (de la relación Cliente_CA) de donde provienen los fondos; destino corresponde a un número de Caja de Ahorro destino

B. Consideraciones generales

B.1. Verificación de la Base de Datos

En el la sección DOWNLOADS/PROYECTOS de la página web, estará disponible para bajar un programa llamado *verificar*. Este programa realiza una verificación sobre la estructura de la base de datos *ya creada*, y muestra un listado con los errores (si los tuviera) que esta presenta con respecto al modelo relacional propuesto en el apendiceA . Este programa debe ejecutarse con el servidor de MySQL corriendo, una vez creada la base de datos. Se recomienda verificar su base de datos con este programa antes de empezar a desarrollar la aplicación. **No se aceptarán proyectos que no pasen correctamente la verificación realizada por este programa**.

B.2. Funciones de MySQL para el manejo de fechas y tiempo

MySQL provee funciones para manipular fechas y tiempo que resultan muy útiles para el desarrollo de este proyecto. En general estas funciones se pueden consultar al servidor mediante la sentencia SQL select y devuelven una tabla con una única celda que contiene el resultado (más información ver sección 12.7 de refman-5.6-en.a4.pdf o sección 12.5 de refman-5.0-es.a4.pdf). A continuación se muestran ejemplos de algunas de las funciones disponibles.

Importante: los ejemplos que se muestran son el resultado de invocar a las funciones desde el cliente *mysql.exe*. El formato de la fecha y hora recuperado desde JAVA a través de JDBC puede variar según el formato de fecha que esté configurado en Windows. **La aplicación entregada deberá funcionar correctamente independientemente de la configuración de fecha y hora de Windows**. Para esto puede utilizar las funciones provistas en la clase *Fechas* contenida en el archivo "*Fechas.java*" de "*ejemplo java-MySQL.zip*" que estará disponible en la página web de la materia.

- ```
  mysql> SELECT CURDATE();
  +------------+
  | CURDATE()  |
  +------------+
  | 2005-09-27 |
  +------------+
  1 row in set (0.03 sec)
  ```
  Devuelve la fecha actual en la forma: año-mes-día
- ```
  mysql> SELECT CURTIME();
  +-----------+
  | CURTIME() |
  +-----------+
  | 13:09:37  |
  +-----------+
  ```

```
1 row in set (0.00 sec)
```

Devuelve la hora actual .

- ```
  mysql> SELECT NOW();
  +---------------------+
  | NOW()               |
  +---------------------+
  | 2005-09-27 13:09:45 |
  +---------------------+
  1 row in set (0.00 sec)
  ```

  Devuelve la fecha y hora actual .

- ```
  mysql> SELECT DAYOFWEEK('2005-09-26');
  +-------------------------+
  | DAYOFWEEK('2005-09-26') |
  +-------------------------+
  |                       2 |
  +-------------------------+
  1 row in set (0.02 sec)
  ```

  El número devuelto representa el día de la semana correspondiente a la fecha dada. 1 = domingo, 2 = lunes, 3 = martes, ... , 7 = sábado.

- ```
  mysql> SELECT TIMEDIFF('13:40:00', '08:05:00');
  +----------------------------------+
  | TIMEDIFF('13:40:00', '08:05:00') |
  +----------------------------------+
  | 05:35:00                         |
  +----------------------------------+
  1 row in set (0.00 sec)
  ```

  `TIMEDIFF(h1, h2)` devuelve la diferencia de tiempo entre dos horas `h1` y `h2`. `h1` debe ser mayor que `h2`, sino devolverá un número negativo.

  `TIMEDIFF` también se puede usar para calcular la diferencia de tiempo entre dos fechas:

  ```
  mysql> SELECT TIMEDIFF('2005-09-27 01:01:01','2005-09-26 23:59:59');
  +-----------------------------------------------------------+
  | TIMEDIFF('2005-09-27 01:01:01','2005-09-26 23:59:59') |
  +-----------------------------------------------------------+
  | 01:01:02                                                  |
  +-----------------------------------------------------------+
  1 row in set (0.00 sec)
  ```

- ```
  mysql> SELECT TIME_TO_SEC('1:00:00');
  +------------------------+
  | TIME_TO_SEC('1:00:00') |
  +------------------------+
  |                   3600 |
  +------------------------+
  1 row in set (0.00 sec)
  ```

  Devuelve la cantidad de segundos que representa el tiempo dado.

B.3. Funciones en MySQL para cifrado de datos

Supongamos que en una base de datos creamos la siguiente tabla para almacenar los usuarios junto con su contraseña o password, y así poder controlar el ingreso al sistema.

```
create table usuarios(
  usuario VARCHAR(30) not null,
  password CHAR(32),
  primary key (usuario)
);
```

Si las contraseñas se almacenan en texto plano y alguien logra acceder a la base de datos, podría recuperar las contraseñas de todos lo usuarios y acceder al sistema.

MySQL provee varias funciones para el cifrado de datos, que permiten almacenar este tipo de información sensible de manera segura (más información ver sección 12.13 de refman-5.6-en.a4.pdf o sección 12.9.2 de refman-5.0-es.a4.pdf). Nosotros utilizaremos la función hash `md5`. Esta función toma como entrada una cadena de texto de cualquier longitud y devuelve una cadena de texto cifrada de 32 caracteres hexadecimales, utilizando el algoritmo MD5 (Message-Digest Algorithm 5). Lo interesante de este algoritmo es que su proceso es irreversible, es decir, a partir de una cadena cifrada no es posible obtener la cadena de texto original.

Por ejemplo, si quisiéramos almacenar un usuario 'u1' con password 'pw1' podríamos hacerlo de la siguiente forma:

```
insert into usuarios values('u1', md5('pw1'))
```

Luego, si consultamos la tabla usuarios podemos ver que el password se almacenó de forma cifrada:

```
mysql> select * from  usuarios;
+---------+----------------------------------+
| usuario | password                         |
+---------+----------------------------------+
| u1      | 6e6fdf956d04289354dcf1619e28fe77 |
+---------+----------------------------------+
```

por lo tanto, si alguien logra acceder a la la base de datos no podrá obtener el password original. Si desde una aplicación queremos validar el ingreso de un usuario al sistema, simplemente tomamos el pasword ingresado por el usuario, le aplicamos la función `md5` y comparamos el resultado con el valor almacenado en la base de datos. Por ejemplo:

```
mysql> select * from  usuarios where usuario='u1' and password=md5('pw1');
+---------+----------------------------------+
| usuario | password                         |
+---------+----------------------------------+
| u1      | 6e6fdf956d04289354dcf1619e28fe77 |
+---------+----------------------------------+
```

si el password introducido por el usuario '`u1`' es incorrecto (distinto de '`pw1`') la consulta anterior no devolverá ningún resultado y de esta forma podemos controlar la autenticidad del mismo.