PRÁCTICA 1

BASES DE DATOS

Curso 2019/2020

CREACIÓN Y CONSULTAS A UNA BASE DE DATOS USANDO JAVA

1. Objetivos Generales

- Crear una base de datos en base a un diagrama E-R e insertar el contenido mediante la creación de un programa inicial de carga de datos.
- Acceder a una Base de Datos y realizar consultas SQL mediante un programa implementado en código Java.
- Manejar el conector JDBC (Java DataBase Conectivity), librería de clases para el acceso a un SGBD y para la realización de consultas SQL desde Java.

2. Objetivos específicos

El alumno será capaz de:

- Realizar conexiones a un SGBD desde un programa Java.
- Realizar consultas simples y complejas SQL manejando las clases adecuadas del conector JDBC.
- Tratamiento de los resultados de las consultas SQL dentro del código Java.
- Manejo de las excepciones en la gestión de consultas.
- Implementación de un sistema basado en el concepto de transacción usando código Java.

3. Metodología

El alumno dispone de la Máquina Virtual del curso en la que se ha configurado la misma para su utilización como gestor (servidor) de una Base de Datos MySQL. Desde el software MySQL Workbench el alumno también puede importar una base de datos cargándola como si fuera un script y ejecutando las sentencias correspondientes, así como realizar las tareas de implementación y consulta que considere oportunas.

La Máquina Virtual del curso está disponible para su descarga en el enlace que se puede encontrar en Moodle. En el *Anexo 1* se describe cómo utilizar este paquete.

Adicionalmente la Máquina Virtual del curso dispone de una *máquina virtual java* y de la herramienta de compilación y ejecución *Eclipse*. Desde la Máquina Virtual del curso y con la herramienta *Eclipse* se implementará el código fuente java pedido en este proyecto. La herramienta Eclipse, instalada en la máquina virtual del curso, dispone en la carpeta "*C:\eclipse\CONECTOR_MYSQL*" del conector JDBC "*mysql-connector-java-5.1.38-bin.jar*" que se deberá usar para la realización de este proyecto. Asimismo, tiene configurada la carpeta "C:\eclipse\proys" como la ubicación por defecto de los proyectos *java*.

Se recuerda que el uso de la máquina virtual para esta práctica, aunque no es obligatorio si es recomendable, y que toda duda que surja por la no utilización de la máquina virtual es responsabilidad del alumno buscar una solución. Además, la corrección de la práctica se llevará a cabo en dicha máquina virtual, de forma que, si surgen fallos por compatibilidad de versiones, la calificación será de cero.

4. Práctica a realizar

Se desea realizar una aplicación Java que gestione con la información de un juego de Pokémon. La base de datos de dicha aplicación tiene que almacenar el número de pokédex, nombre, descripción y sprite de cada especie. También se deberá guardar desde qué especie evoluciona otra especie dada.

Se almacenan también los ejemplares de cada especie que se capturan en el juego, los cuáles se identifican mediante el número de pokédex de la especie y un contador de encuentro correlativo para dicha especie. Además, de cada ejemplar se almacena su apodo, el sexo, el nivel y si ha sido infectado o no por un nuevo virus descubierto recientemente y que está creando una pandemia entre los Pokémon: el coronapokerus.

También se guardará información sobre los ataques, con su identificador, su nombre y su potencia. Además, tanto los ataques como las especies tienen uno o varios tipos, de los que se almacena también el identificador y el nombre.

Una especie puede aprender determinados ataques al llegar a un determinado nivel y se debe almacenar esta información, así como la de qué ataques en concreto conoce cada ejemplar. La Figura 1 muestra el diagrama E-R que modela la base de datos mencionada.

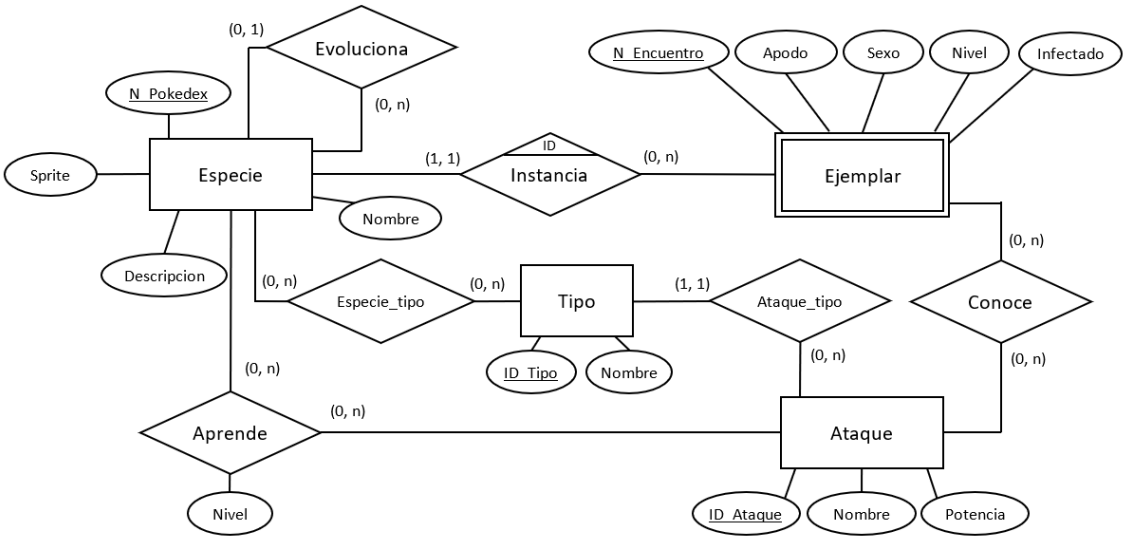


Figura 1. Diagrama entidad relación de la base de datos "Pokemon".

Junto a este enunciado encontrarás cuatro elementos:

- *pokemon.sql*: incluye la definición del esquema de la base de datos (y su borrado si ya existe para resetearlo, ¡usar con precaución!), así como la creación de varias tablas y la inserción de datos en ellas. Este script debe importarse o ejecutarse desde el Workbench antes de comenzar a desarrollar la aplicación.
- *pokemon*: paquete de Java que incluye seis clases. La clase *PokemonDatabase* contiene el esqueleto de los métodos a implementar y es el único fichero que debe modificarse. El estudiante debe entregar como resultado de esta práctica únicamente el fichero .java que

implementa esta clase, completando con el código todo lo que se pide. Por otra parte, el paquete contiene una clase *Main*, que implementa un menú para probar la funcionalidad de la aplicación y cuatro clases que emulan las estructuras de datos de cuatro tablas.

Importante: queda prohibido modificar la definición de cualquier método ya existente en el fichero. Todos ellos deben hacer un correcto tratamiento de las excepciones. Si se necesita crear algún método adicional, éste debe ser privado.

- *aprende.csv*: fichero CSV que incluye la lista de ataques que cada especie aprende a un determinado nivel y que debe cargarse en el método *loadAprende*. Cada línea de este fichero tiene el formato "*N_Pokedex;ID_Ataque;nivel*".
- *conoce.csv*: fichero CSV que incluye la lista de ataques que cada ejemplar concreto conoce y que debe cargarse en el método *loadConoce*. Cada línea de este fichero tiene el formato "*N_Pokedex;N_Encuentro;ID_Ataque*".

Con todo ello, se deben implementar los siguientes métodos en Java (además de añadir el código que fuera necesario en cualquier otra parte del fichero):

1. `connect` [0.25 puntos]: este método debe implementar la apertura de la conexión con la base de datos. El método devuelve *true* si la conexión está abierta tras ser llamado (porque se abra o ya estuviera abierta previamente) y *false* en caso contrario. Se debe llamar a este método cuando se vaya a realizar la conexión en otras partes del código. Nunca debe abrir varias conexiones.
2. `disconnect` [0.25 puntos]: este método debe implementar el cierre de la conexión con la base de datos. El método devuelve *true* si se ejecuta sin errores y *false* si ocurre alguna excepción. No es necesario llamar a este método en el código, ya está llamado donde corresponde en la clase *Main*.
3. `createTableAprende` [0.5 puntos]: este método debe implementar la creación de la tabla *aprende*. Prestar atención a cualquier restricción del enunciado y nombrar las claves foráneas usando exactamente el mismo nombre que la primaria a la que referencia (la tabla debe llamarse exactamente *aprende*). El método devuelve *false* si la tabla no se ha podido crear (por algún tipo de error o porque ésta ya existía) y *true* si la tabla se ha creado correctamente.

   Al margen de las claves foráneas, la columna para almacenar el nivel debe llamarse exactamente *nivel* y debe ser de tipo entero.
4. `createTableConoce` [0.5 puntos]: este método debe implementar la creación de la tabla *conoce*. Prestar atención a cualquier restricción del enunciado y nombrar las claves foráneas usando exactamente el mismo nombre que la primaria a la que referencia (la tabla debe llamarse exactamente *conoce*). El método devuelve *false* si la tabla no se ha podido crear (por algún tipo de error o porque ésta ya existía) y *true* si la tabla se ha creado correctamente.
5. `loadAprende` [1.5 punto]: este método debe insertar en la base de datos los ataques que aprende cada especie y que están contenidos en el fichero *aprende.csv*. Para ello se dispone del método `readData` (ya implementado) de la clase *Aprende*, el cual procesa el fichero CSV y retorna un ArrayList con elementos de la clase *Aprende* (también

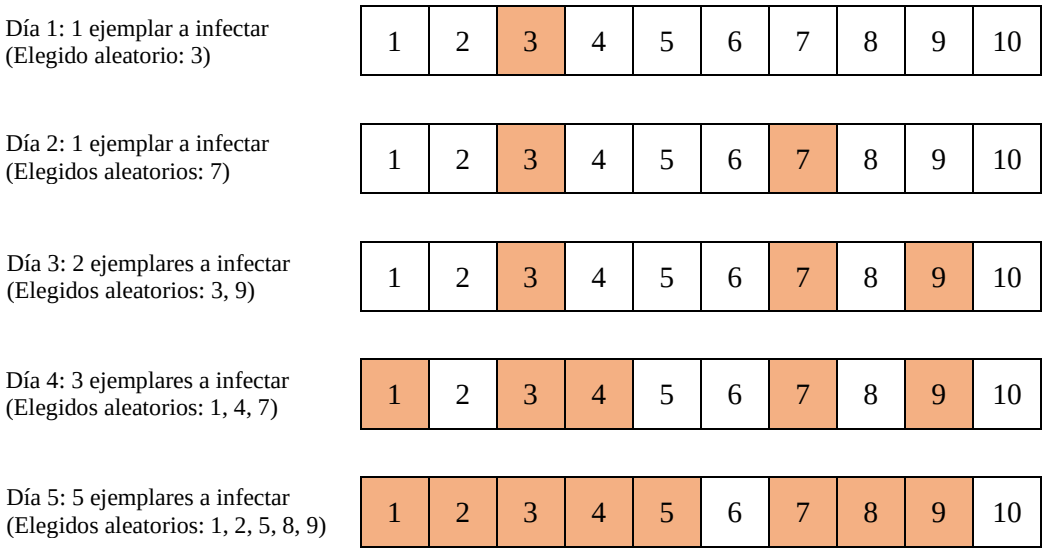
implementada). Cada inserción debe ser tratada como una transacción separada del resto. El método debe retornar la cantidad de elementos insertados en la tabla.

6. `loadConoce` [2 puntos]: este método debe insertar en la base de datos los ataques que conoce cada ejemplar y que están contenidos en el fichero *conoce.csv*. Para ello se dispone del método `readData` (ya implementado) de la clase *Conoce*, el cual procesa el fichero CSV y retorno un ArrayList con elementos de la clase *Conoce* (también implementada). Obligatoriamente, debe ejecutarse la creación y carga de todos los datos como si fuera una única transacción, de tal forma que cualquier fallo intermedio de lugar a deshacer por completo los cambios anteriores. El método debe retornar la cantidad de elementos insertados en la tabla.
7. `pokedex` [1 punto]: este método debe consultar en la base de datos la lista de todas las especies e introducir todos los datos de éstos en un ArrayList de la clase *Especie* (ya implementada en el código), el cuál debe ser retornado por el método. Si algún campo es nulo en la base de datos, no hay que hacer tratamiento especial de éste, sino almacenar el valor que los métodos get de ResultSet nos proporcionan. Si no hay especies almacenadas, debe retornarse un ArrayList vacío. En cambio, si se produjera alguna excepción, el método debe retornar *null*.
8. `getEjemplares` [1.5 puntos]: este método debe consultar en la base de datos la lista de todos los ejemplares usando orden ascendente del número de pokédex y, en caso de empate, usar como criterio de desempate el orden ascendente del número de encuentro. Debe introducir todos los datos de éstos en un ArrayList de la clase *Ejemplar* (ya implementada en el código), el cuál debe ser retornado por el método. Si algún campo es nulo en la base de datos, no hay que hacer tratamiento especial de éste, sino almacenar el valor que los métodos get de ResultSet nos proporcionan. Si no hay especies almacenadas, debe retornarse un ArrayList vacío. En cambio, si se produjera alguna excepción, el método debe retornar *null*.
9. `coronapokerus` [2 puntos]: este método se encarga de liberar la pandemia del coronapokerus. Tiene como parámetros el número de días que la pandemia debe durar y un ArrayList sobre los ejemplares que la pandemia se extenderá (este ArrayList se extrae en la clase *Main* con el método del punto anterior). El virus debe infectar cada día de forma aleatoria a tantos ejemplares como hubiera ya infectados el día anterior (excepto el primer día, que infecta siempre a un ejemplar). Para seleccionar el ejemplar aleatorio al que se va a infectar, debe usarse el método `ejemplarRandom` de la clase Ejemplar, al que se le pasa un ArrayList de ejemplares y retorna de forma aleatoria uno de ellos. No deben realizarse más llamadas a este método de las necesarias. A la hora de hacer esta selección, es indiferente que el ejemplar ya esté infectado, dicho ejemplar ya infectado cuenta dentro de los que habría que infectar el nuevo día.

   A la hora de volcar los datos a la base de datos, cada día debe representar una transacción independiente y, en caso de que ocurra una excepción, deben deshacerse todos los cambios de ese día y terminar la ejecución del método en ese punto.

   El método debe retornar la cantidad final de ejemplares infectados.

   A continuación se muestra un ejemplo del proceso de infección con 5 días (con un total de 12 llamadas realizadas al método `ejemplarRandom`):



10. `getSprite` [1 punto]: el método debe obtener la imagen del sprite del pokemon cuyo número de pokedex se pasa como parámetro. La imagen debe almacenarse en la ruta indicada por el parámetro *filename*. Debe retornar *true* si la imagen existía en la base de datos y se ha almacenado correctamente y *false* en caso contrario.

El programa debe conectarse cuando se vaya a realizar una consulta/operación contra la BD por primera vez (no al arrancar el programa) y no debe desconectarse hasta que el programa finalice (de esto último se encarga ya la clase Main).

5. Consideraciones sobre el paquete Java *pokemon*

El paquete de Java *pokemon* contiene los métodos necesarios para ser ejecutado, ya que la clase *Main* contiene el método *main*, que ejecuta el menú del programa. El menú está preparado para que se ejecuten las diferentes operaciones que se pueden escoger (de la 0 a la 5) y que, tras ello, vuelva a mostrarse el menú. Sólo se sale del menú (y del programa) escogiendo la opción 0, que es la de salir. Está prohibido modificar esta clase.

El paquete tiene implementadas también las clases *Especie*, *Ejemplar*, *Aprende* y *Conoce*, que tienen las estructuras para almacenar la misma información que las tablas que se llaman igual y disponen de sus constructores y métodos *getter* y *setter*. Las clases *Aprende* y *Conoce* tienen implementado un método que debe ser usado para obtener datos de los ficheros aprende.csv y conoce.csv, respectivamente: *readData*. Estos métodos devuelven un ArrayList con los ataques que una especie aprende a un determinado nivel y los que un ejemplar determinado conoce respectivamente. Además, la clase *Ejemplar* contiene un método que debe usarse para seleccionar un ejemplar de forma aleatoria entre los pasados como parámetro en un ArrayList. Está prohíbo modificar cualquiera de estas clases (de hecho, no deben entregarse)

La clase *PokemonDatabase* contiene las cabeceras de los métodos que se pide implementar, con lo que el estudiante simplemente debe rellenar el código en dichos métodos. Está prohibido modificar la definición de dichos métodos, pero pueden definirse tantos métodos privados nuevos como se deseen. El fichero que contiene esta clase es lo único que se debe entregar.

6. Evaluación y otras indicaciones

Es importante que se cumplan los requisitos establecidos en la práctica, incluyendo:

- El método connect() debe ser el encargado de: cargar el driver y realizar la conexión. Es obligatorio que se implemente esta funcionalidad dentro de dicho método. Parámetros:
    - Servidor: localhost:3306
    - Usuario: pokemon_user
    - Password: pokemon_pass
    - Nombre base de datos: pokemon

- Que sean los métodos ya dados los que, al menos, se encarguen de proporcionar el resultado final. El estudiante puede generar otros métodos adicionales si lo considera relevante, pero el método asignado a cada ejercicio debe devolver el resultado.

- El fichero de la clase Java a entregar debe llamarse obligatoriamente "PokemonDatabase.java", tal y como se entrega. Es decir, debe usarse en realidad ese fichero, no se debe crear uno nuevo. No debe cambiarse tampoco el nombre del paquete. Solo puede entregarse dicho fichero .java. En caso de necesitar crear objetos extra para el procesamiento de datos (aunque no se considera necesario), deben hacerse como clases inner[1].

- La práctica se evaluará de 0 a 10 mediante un corrector automático. Esto quiere decir que los métodos deben funcionar perfectamente y todos los formatos deben ser acordes al presente guion. A la hora de la corrección se usarán tanto los datos especificados aquí, como otros datos para la comprobación de la corrección de las soluciones. Si un método no pasa la prueba automática, se calificará con 0 puntos. Esta corrección automática dará una nota preliminar de la práctica, que puede ser inferior si se dan los supuestos que se enumeran a continuación.

    - **Limpieza y claridad del código** que, además, debe gestionar correctamente de errores, estar optimización, contener comentarios, etc.: no cumplir con estos aspectos puede suponer la deducción de hasta 1.5 puntos.
    - **Fichero entregado con nombre incorrecto o entrega de ficheros adicionales:** -0.5 puntos.
    - **Conexión con un usuario que no sea el dado:** -2 puntos (si es con root), -1 punto (si es con otra combinación de usuario/contraseña inválida).
    - **Realización de más de una conexión/realización de conexión en momento inoportuno:** -1 punto.
    - **No usarse los cierres de las estructuras necesarias:** -0.5 puntos.
    - **No usar PreparedStatement cuando haya parámetro o usarlo incorrectamente (usando concatenaciones):** -3 puntos.
    - **Creación de tablas sin claves foráneas:** -2 puntos.

[1] https://www.tutorialspoint.com/java/java_innerclasses.htm

- **Modificación de una parte prohibida de los ficheros:** supone la calificación automática con un 0.
- **No identificar correctamente los tipos de errores:** -0.5 puntos.
- **Lanzamiento de las excepciones hacia arriba:** para esto es necesario modificar las cabeceras de los métodos y supone la calificación automática con un 0.
- **La corrección se realizará sobre la máquina virtual.** El hecho de que el fichero no funcione sobre dicha máquina supondrá la calificación automática con un 0.
- Otros supuestos: Los profesores pueden aplicar otras penalizaciones en caso de encontrar errores diferentes a los aquí descritos. Quedará a discreción del profesorado la penalización a aplicar en cada caso.

- La calificación final de la práctica no superará en ningún caso los 10 puntos.
- Los profesores se reservan el derecho de citar a cualquier grupo a defender la práctica si lo considerara necesario.

7. Entrega de la Práctica y resolución de dudas

El grupo de prácticas deberá **entregar mediante la plataforma Moodle en una tarea habilitada para ello un único fichero comprimido (*.zip, *.rar) cuyo nombre sea (Grupo_N_JDBC.rar) (donde N es el número de grupo correspondiente) que contenga:**

1. Clase PokemonDatabase.java con la implementación solicitada.
2. Opcionalmente un documento en PDF que contenga comentarios acerca de los problemas surgidos al hacer la práctica o cualquier otro comentario que los alumnos estimen oportuno.

La práctica debe ser entregada por **sólo uno de los miembros del grupo** (preferiblemente el representante).

La fecha límite para la entrega de esta práctica es el jueves **30 de abril de 2020 a las 23:55**.

8. Resolución de dudas

Las dudas deben plantearse en primer a instancia a través del foro habilitado para dicho fin en Moodle. Si fuera necesario concertar una tutoría, debe enviarse un correo electrónico al profesor Antonio Jesús Díaz Honrubia (antoniojesus.diaz@upm.es).

A1. 1 Paquete XAAMP

Con el fin de facilitar la tarea de la realización de las prácticas propuestas en la asignatura de Bases de Datos se proporciona el software "*PAQUETE XAMPP PRÁCTICAS SQL*". Este paquete es una versión del software *XAMPP* de libre distribución (GNU General Public License).

Este paquete software contiene un gestor de *Bases de Datos Mysql* que se propone para hacer las prácticas de la asignatura. La ventaja de utilizar este paquete es que el alumno no tiene que instalar ningún servicio ni ninguna aplicación en su sistema.

Este paquete aparece integrado en la Máquina Virtual del curso. En este caso el paquete ya está configurado y listo para ser usado.

En este momento alumno podrá utilizar las aplicaciones incluidas en el paquete *XAMPP*. Para la práctica propuesta el alumno deberá pinchar en primer lugar el botón "*start*" de la aplicación "*Apache*". Si todo ha ido bien deberá aparecer la palabra "*running*" resaltada de color verde a la altura del nombre de la aplicación como se aprecia en la figura A4. A continuación deberá pinchar el botón "*start*" de la aplicación "*Mysql*". Si todo ha ido bien deberá también aparecer la palabra "*running*" resaltada de color verde a la altura del nombre de la aplicación como se aprecia en la figura A1.1.

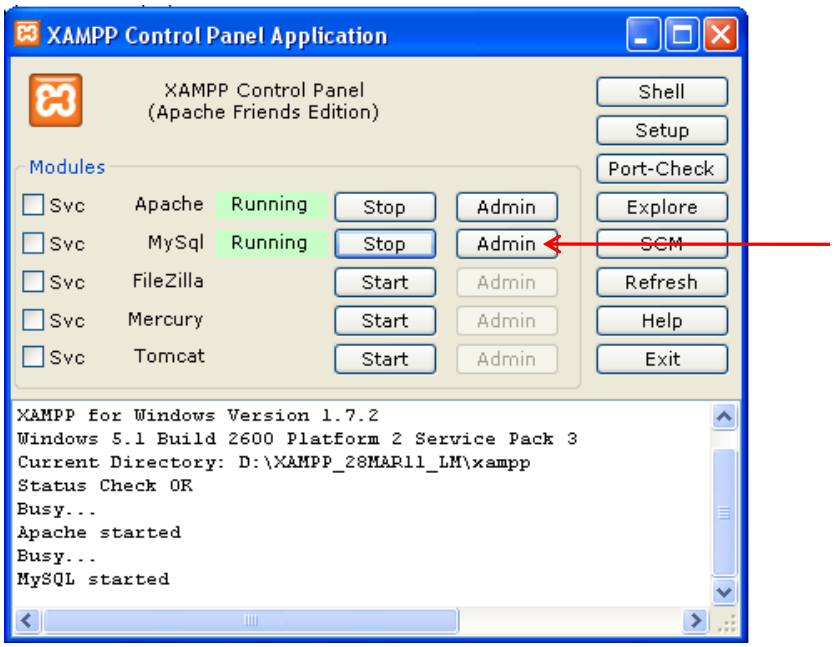


Figura A1.1

Ahora, el alumno puede pinchar el botón "*Admin*" del panel de control del paquete *XAMPP* (ver Figura A4). De este modo el alumno accede al entorno denominado "*phpMyAdmin*" el cual es una aplicación web de acceso administrativo al gestor de Base de Datos *MySql*. Desde este entorno (ver Figura A1.2) el alumno puede por ejemplo exportar una base de datos, ver el diagrama de esquema de una Base de Datos, crear tablas, hacer consultas de distintas tablas, etc.

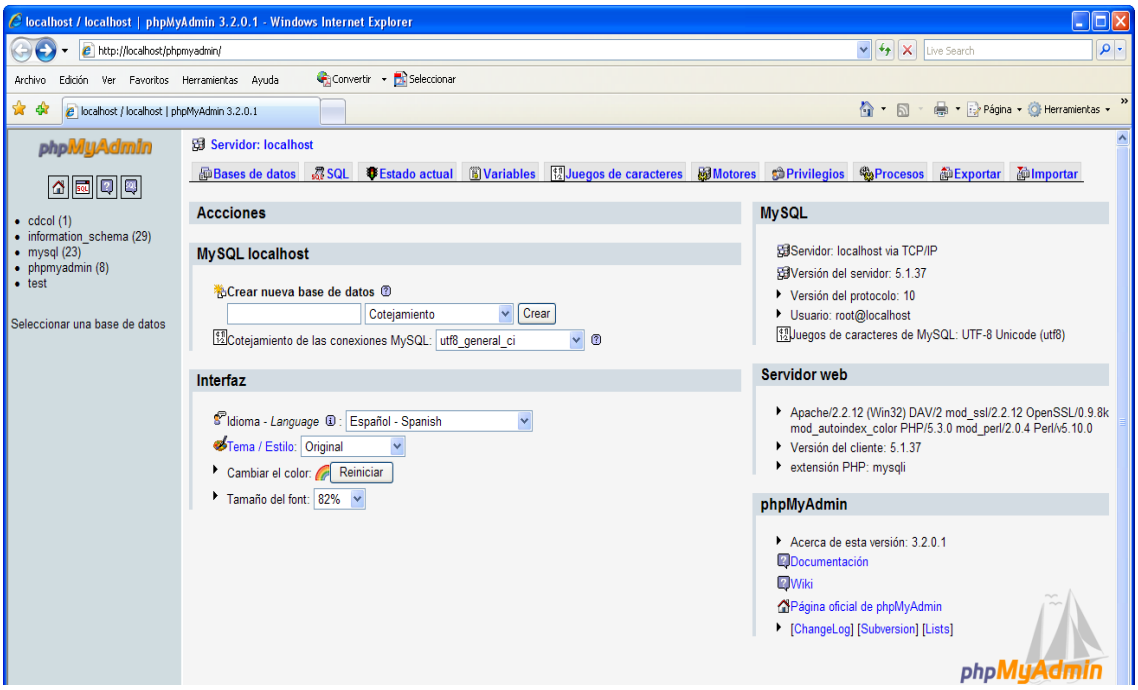


Figura A1.2: Entorno Administración "*phpMyAdmin*"

Se recomienda, no obstante, que el alumno utilice un cliente, por ejemplo, el *"MySQL Workbench"* para conectarse al Servidor *MySQL* y realizar consultas DDL (Data Definition Language) y DML (Data Manipulation Language), dejando el entorno "*phpMyAdmin*" para tareas de administración (P.e. exportar una Base de Datos y ver el diagrama de esquema de una Base de Datos. (ver Figura A1.3).

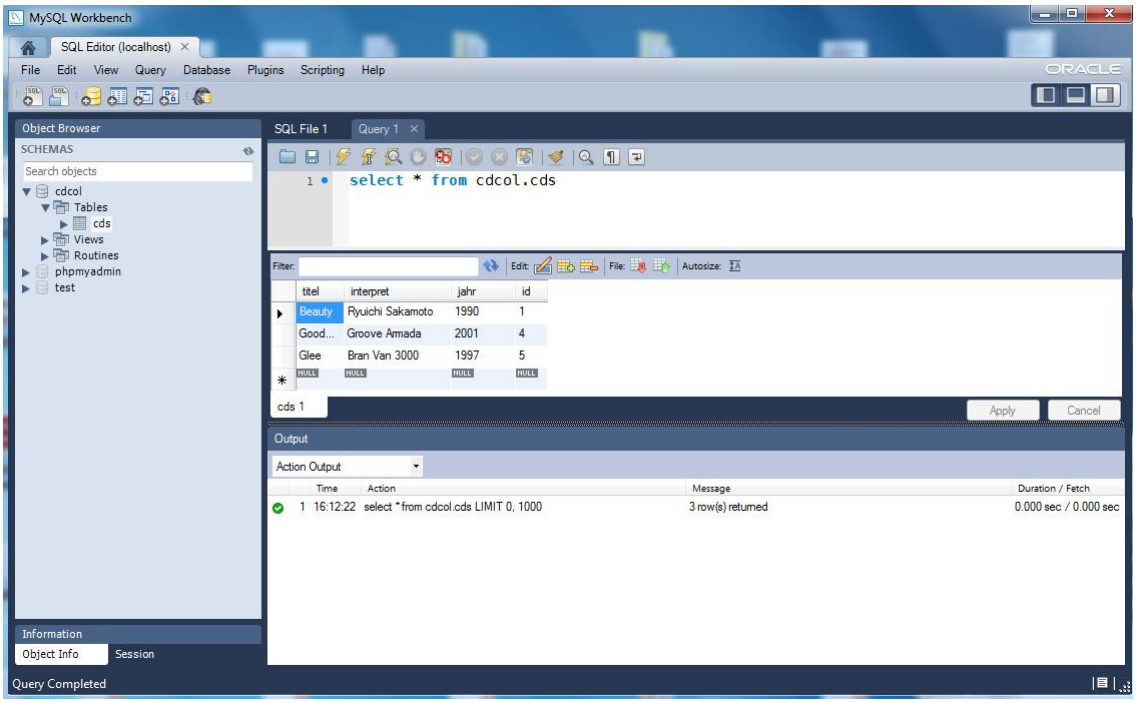


Figura A1.3: Cliente *Workbench*