Dedicated computer 🡪 Únicamente se va a instalar el servidor SQL, nada más. Cuando se conectas a Amazon lo que haces es conectarse a un servidor, es decir, a otro ordenador que está nosédonde. Aquí usa la memoria que quiere porque solo va a estar él.

Server computer 🡪 Ordenador donde hay instalado más de un servidor. Por tanto va más tranquilito a la hora de usar memoria.

Development computer 🡪 La que vamos a coger porque es la más sencilla, es la que trabajas en tu PC con otras aplicaciones.

PUERTO ES TÍPICA PREGUNTA DE EXAMEN EN LA PARTE DE PYTHON 3306 ES EL PUERTO QUE UTILIZA MYSQL

En ocasiones podrías cambiar el puerto porque el puerto ya está en uso o porque pueda no interesarte que la gente sepa que tienes instalado un servidor de base de datos en tu máquina. “Esconder” la dirección de tu puerto puede ser muy importante porque en estos servidores se puede guardar información delicada, con el objetivo de dificultar que otra gente lo encuentre. Esto lo dificulta pero no lo impide, porque hay herramientas que permiten dado una máquina ver todos lo puertos posibles a los que están conectados.

Si tenemos una versión de mySQL antigua ponemos el legacy authentication para poder tener compatibilidad con las configuraciones antiguas, si no es el caso (no es el caso) ponemos el de arriba (SPEA) que es más seguro dado que permite encriptar las contraseñas.

Cuando guardemos información de contraseñas nunca se debe guardar el dato plano sino que hay que cifrarlo.

Una vez hemos entrado al workbench tenemos una instancia según entramos, y vemos el onmbre del usuario que es root, y un **localhost**, que es mi máquina.

Si te vas a servicios en la barra de b´suqueda de Windows puedes buscar en la lista MySQL80, le das a botón derecho y a suepnder/apagar/terminar. Así lo cierras. Si abres el workbecnh se te abre pero evidentemente no está conectado. Si te vas a schemas cuando pone not connected es que no estás conectado. La diferencia entre el root y el otro añadido a un puerto al azar, como te has conectado previamente se han guardado ciertas credenciales que te permiten hacer cositas. Por ejemplo puedes irte a startup/shutdown y arrancar el servidor desde ahí.

Hay que tener cuidadín y ver que SIEMPRE estás conectado, por lo que para arrancar el servidor lo más fácil es ir a administración --> startup/shutdown.

**TERMINOLOGIA**

Tabla 🡪 Pos eso que los datos se guardan en tablas compuestas de columnas y filas, formadas por campos.

MySQL usa un modelo de bases de datos relacional, evitando que haya contenido duplicado y el contenido se relaciona por llaves, llave primaria y foránea.

Llave foránea 🡪 Llave que se agrega como columna a la tabla que queremos relacionar. Ej: vategoriaID 🡪idCategoria. Relacionar las tablas con llaves permite que no se repitan los datos, simplemente puedes crear una relacion entre 2 tablas en vez de guardar todo 2 veces.

**TIPOS DE DATO**

Numéricos 🡪 INT(número entero), TINYINT, BIGINT, FLOAT(se usa para almacenar precios y demás datos decimales)

String 🡪 VARCHAR(más común, hay que especificar cantidad máxima de caracteres), TEXT, CHAR

Fecha y hora 🡪 DATE(Fecha), DATETIME(Fecha y hora), TIME (Hora), TIMESTAMP(otro formato de fecha y hora)

Otros 🡪 BLOB(almacena archivos), JSON(almacena muchos datos distintos)

**CÓDIGO**

Por lo general se escribe en mayúsculas, las líneas se separan con punto y coma (;)

**SHOW DATABASES** 🡪Te enseña las bases de datos existentes

**CREATE nombre 🡪** Te crea una base de datos que se llama “nombre”

**USE nombre 🡪** Usas la base de datos que se llama “nombre”

**SHOW TABLES 🡪** Te enseña las tablas que están en la base de datos en la que estés

**CREATE TABLE nombre 🡪**Crea una tabla en la base de datos de nombre “nombre”

**EJEMPLO: CREATE TABLE ejemplo (id INT(11) NOT NULL AUTO\_INCREMENT, nombre VARCHAR(60) NOT NULL, apellido VARCHAR(60) NOT NULL, dirección VARCHAR(60), PRIMARY KEY (id));**

-id 🡪Título de la primera columna, que será un identificador de cada uno de los elementos de la tabla.

-INT 🡪Defines que ese id estará formado por un número entero

-(11) 🡪 La longitud de ese número entero no será mayor a 11

-NOT NULL 🡪 A la hora de rellenar la tabla este campo no puede ir vacío

-AUTO\_INCREMENT 🡪 Cada vez que se mete una fila nueva, el valor del ID se ve incrementado en 1, así es más cómodo meter los datos

**CREATE TABLE ejemplo (id INT(11) NOT NULL AUTO\_INCREMENT, nombre VARCHAR(60) NOT NULL, apellido VARCHAR(60) NOT NULL, dirección VARCHAR(60), PRIMARY KEY (id));**

-PRIMARY KEY (id) 🡪 Especificas que la key primaria es la columna id

-nombre 🡪 Título de la siguiente columna

-VARCHAR 🡪 carácter variables, es decir, es un string pero de cualquier naturaleza

-(60) 🡪 Como mucho será de longitud 60

Puedes poner =, <>(distinto), <, >, <=, >=. Si quieres buscar una palabra por ejemplo pones categoría=’palabra’

**DESCRIBE nombretabla** 🡪Te muestra los títulos de la columna de la tabla seleccionada y información sobre cada una de ellas

**CRUD**

**INSERT INTO nombretabla (campos que quieres rellenar) VALUES (‘primeracolumna’, ‘segunda’, ‘tercera’) 🡪** Primero le dices qué fila quieres rellenar y las columnas que quieras rellenar y después los elementos que quieres poner en la primera fila para rellenarla.

**Ejemplo: INSERT INTO holi (nombre, apellido, dirección)**

**VALUES (‘Juan’, ‘De La Torre’, ‘Avenida De La Piruleta 29012’);**

**SELECT \* FROM nombretabla 🡪** Si pones el asterisco te muestra todas las columnas de la tabla especificada, si quieres seleccionar solo unas columnas cambias el \* por los campos que quieras separados por una coma (,)

**SELECT \* FROM nombretabla WHERE ID = 2; 🡪** En este caso únicamente te enseña la fila de la tabla que tenga el id = 3

**SELECT \* FROM nombretabla LIMIT 1 🡪 En vez de enseññarte la table entera solo te enseña las filas que especifiques (en este caso 1)**

**UPDATE nombretabla SET nombretitulocolumna = ‘nuevo nombre’, nombretitulocolumna =’nuevonombre’ WHERE id = 1;**

**EJEMPLO:** UPDATE holi SET nombre =’Pablo2’, dirección=’Av. De la Piruleta 34’, WHERE id = 1;

**DELETE FROM nombretabla WHERE id =1; 🡪**Te elimina la fila con id = 1

**ALTER TABLE nombretabla ADD nuevotitulocolumna TIPOVARIABLE(longitud) 🡪** Creas un nuevo título de columna a la tabla que selecciones

**ALTER TABLE nombretabla DROP COLUMN nombrecolumna** 🡪 Eliminas la columna que hayas especificado de la tabla que hayas dicho

Puede darse el caso de que partas de una base de datos ya establecida, en este caso simplemente deberás para hacer text mining abrir el script desde mySQL y desde el menú de la izquierda eliges el nombre de esa base de datos y le das doble click para abrirla (recomendable ejeutar el script). A partir de ahí, como ya has seleccionado la base de datos y la has ejecutado y estás en ella puedes abrir un nuevo script y empezar a hacer cosas.

**SELECT \* FROM nombretabla ORDER BY columna** 🡪Este comando te permite ordenar (por defecto en orden ascendente) los elementos de la tabla en función de la columna que elijas, en este caso “columna”. Por defecto es por orden ascendete (**ASC**) pero puede ser también por orden descendente (**DESC**).

**SELECT \* FROM nombretabla ORDER BY columna ASC** 🡪 Orden ascendente

**SELECT \* FROM nombretabla ORDER BY columna DESC 🡪** Orden descendente

**SELECT COUNT(id), columnaquequierascontar FROM tabla GROUP BY columnaporlaquequieraagrupar ORDER BY COUNT(id) DESC 🡪** Esto lo que indica es que vas a contar el número de veces que se repiten los elemento de una columna, y te va a mostrar cada uno de los elementos de la columna no repetidos, y el número de veces que se repite cada uno, y todo ello ordenado de forma descendente en función del número de veces que se repiten.

EJEMPLO: **SELECT COUNT(id), fecha FROM reservaciones GROUP BY fecha ORDER BY COUNT(id) DESC 🡪** le indicas con SELECT COUNT(id) que tiene que contar, luego de indicas con la coma y fecha que la columna en la que vas a ver las veces que se repiten es en fecha, después con FROM reservaciones le indias en qué tabla tiene que mirar, después le indicas con GROUP BY fecha que vas a agrupar los datos por su fecha (como es lógico), después con ORDER BY COUNT(id) DESC le indicas que te lo ordene de menor a mayor en función de las veces que se repita cada fecha.

**MAX(columna) 🡪** Funciona exactamente igual que count(columna), pero te muestra únicamente la fila de la columna indicada cuyo valor sea máximo.

**MIN(columna) 🡪** Funciona exactamente igual que count(columna), pero te muestra únicamente la fila de la columna indicada cuyo valor sea mínimo.

**SUM(columna) 🡪** Funciona exactamente igual que count(columna), pero te muestra el sumatorio de los elementos de la columna que especifiques.

**AVG(columna) 🡪** Funciona exactamente igual que count(columna), pero te muestra únicamente la media de los elementos de la columna especificada.

**SELECT \* FROM tabla1, tabla2 🡪** Te fusiona 2 tablas a lo bruto, te pone las columnas de la primera y pegado a ello las columnas de la segunda.

**SELECT \* FROM tabla1 INNER JOIN tabla2 ON categoriacomuntabla1 = categoriacomuntabla2 🡪** Seleccionas 2 tablas que deben tener una categoría en común, la cual indicas con el ON, y te crea una nueva tabla que es fusión de las 2 anteriores

***EJERCICIO***

Dadas las tablas cateogoria y platillos, indicar el número de: desayunos, cenas, bebidas, bebidas alcohólicas… distintas que tienen.

SOLUCION: **SELECT COUNT(platillos.id), categoria.nombre FROM platillos INNER JOIN categoria ON categoria.id = platillos.categoriaId GROUP BY categoria.nombre ORDER BY (platillos.id);** 🡪 Primero haces el SELECT COUNT(platillos.id), en este caso tienes que especificar al table de la que coges el id porque trabajs ahora con 2 tablas. Después con el coma categoría.nombre seleccionas lo que quieres contar que en este caso está en la segunda tabla por lo que hay que especificarlo, y con FROM la tabla 1, que en este caso es platillos indicas la primera tabla. Después haces INNER JOIN tabla2 (categoría) ON tabla1.categoriacomun1 = tabla2.categoriacomun2 (platillos.categoriaId = categoría.id). Despues GROUP BY cateogria.nombre para agrupar por los nombres de plato y por último SORT BY COUNT(platillos.id) DESC para ordenar de menor a mayor.

**SELECT DISTINCT categoría FROM tabla 🡪** Te enseña los valores no repetidos de la categoria que le indiques de la orden especificada, pero no te los cuenta ni nada, solo te enseña los valores no repetidos.

Estos comandos se pueden combinar entre ellos, por ejemplo puedo ponerle al anterior un ORDER BY

**SELECT \* FROM tabla WHERE categoría IS NULL 🡪** Te enseña los elementos de la categoría que no tengan valor, es decir, que tengan valor nulo

**SELECT \* FROM tabla WHERE categoría IS NOT NULL** 🡪 Al revés del IS NOT NULL

**SELECT \* FROM tabla WHERE categoría BETWEEN 100 AND 200 🡪** Aplicas un filtro, de forma que únicamente te enseña los valores comprendidos en el rango marcado de la categoría y tabla especificadas. Incluye los extremos.

**SELECT \* FROM tabla WHERE categoría LIKE ‘palabra’ 🡪** Te busca la palabra que está entre comillas, pero te busca EXACTAMENTE ESA PALABRA, si quieres algo parecido te toca usar operadores lógicos, para indicar “lo que sea después o antes de la palabra especificada” tienes que poner un %. Es decir:

**SELECT \* FROM tabla WHERE categoría LIKE ‘palabra%’ 🡪** El % indica que después de café puede ir cualquier cosa. SI pongo el % antes sería “que acabe con café”.

**SELECT \* FROM tabla WHERE categoría LIKE ‘palabra\_’ 🡪** La \_ indica que después de café puede ir un único caracter.

**SELECT \* FROM tabla WHERE categoría NOT LIKE ‘palabra’ 🡪** Igual que LIKE pero al revés, evita las que coincidan.

**SELECT \* FROM tabla WHERE CONCAT(categoria1, ‘ ‘ , categoria2) LIKE ‘%palabracategoria1 palabracategoria2%’ 🡪** Se usa cuando quieres buscar un elemento que pertenece a distintas columnas, para ello usas **CONCAT( )**.

**CONCAT (categoria1,’ ‘, categoria2,’ ‘,categoria3…) 🡪**Se usa para idicar que vas a usar 2 o más categorías, puede ir después de SELECT para que te enseñe las cateogrias determinadas, después del WHERE para que busque en esas 2…

**EJEMPLO:**

**SELECT CONCAT(nombre,' ', apellido) as 'nombre y apellido' FROM reservaciones WHERE CONCAT(nombre,' ',apellido) LIKE ('%Ana Preciado%'); 🡪** Buscas en la tabla reservaciones el nombre Ana Preciado, pero como buscas e 2 columnas tienes que usar el CONCAT( ) después del WHERE. Para que el output sea solo el nombre y apellido pones un CONCAT( ) también después del SELECT y para que no te ponga en la categoría ‘CONCAT(nombre, apellido) pones el **as** **‘Nombre y apellido’** para que salga bonito como nombre y apellido.

**SELECT \* FROM tabla WHERE categoría IN(condicion1, condicion2, condicion3…) 🡪**Te muestra las filas de la table que cumplen en la categoría especificada las condiciones que has puesto en el IN( ).

**DROP nombretabla 🡪** Elimina el contenido de la tabla pero no la tabla

**IF EXISTS categoría 🡪** Haces un condicional si existe la categoría supongo, yo creo que simplemente pones IF y luego el comando que sea

**SELECT \* FROM nombretabla LIMIT 1 🡪** Únicamente te enseña la primera fila, si en vez de 1 pones 50 pues te enseña los primeros 50, es como hacer un fetchone(), fetchall(), fetchmany)

**SUBCONSULTAS**

Las subconsultas consisten en hacer una búsqueda en una tabla con un elemento que obtenemos a partir de la búsqueda en otra tabla. Para ello escribimos la subconsulta donde buscaríamos el elemento en la primera tabla, y hacemos dicha subconsulta entre paréntesis. Es decir:

**SELECT \* FROM tabla1 where id= (select id from tabla2 where nombre=’Pedro’)**

La desventaja de utilizar el = es que solo nos sirve cuando la subconsulta aporta un único resultado, sin embrago si aportara más no nos serviría el =, tendríamos que utilizar otro operador que es **IN**. Es decir:

**SELECT \* FROM tabla1 WHERE id IN (SELECT id FROM tabla2 WHERE precio > 37)**

Se pueden anidar todas las subconsultas que quieras, eso sí, cuidado que el output de la subconsulta menor sea un input válido en la subconsulta de mayor rango, porque si no no te va a enseñar nada. Las subconsultas, como es lógico, se ejecutan de dentro hacia fuera.

Las subconsultas pueden usarse con cualquier comando, por ejemplo:

**INSERT INTO tabla (SELECT 5, ‘Roberto Córdoba’, direccion, ‘6000000011’ FROM table WHERE nombre = ‘Lucía García’) 🡪** En este caso, lo que quieres hacer es añadir una nueva fila a la tabla, la cual tiene id 5, el nombre es Roberto, y el teléfono 600011, pero la dirección es la misma que Lucía García, por lo tanto hacemos un SELECT entre paréntesis para meter todos los elementos, y a la vez hacer la subconsulta. (No es como has aprendido a hacerlo tú, pero es lo que hay).

METER NUEVAS FILAS EN UNA TABLA USANDO UNA SUBCONSULTA (SE HACE CON SELECT)

Otro ejemplo es con DELETE FROM:

**DELETE FROM tabla WHERE columna1 IN (SELECT columna1 FROM tabla2 WHERE columna2=’%a%’)**

**IMPORTANTE**

**Si queremos usar count(), max(), min() o lo que sea como elemento para comparar, tenemos que poner el count() en la zona de select y con un AS le cambiamos el nombre, y cuando queramos usar ese valor en vez de poner WHERE ponemos HAVING. Si queremos por ejemplo coger los valores mayores a la media, como hay varios outputs que vienen de la subconsulta, si ponemos columna >(SELECT…) nos va dar error, por lo que ponemos ANY, es decir, columna > ANY(SELECT…)**

**Por ejemplo:**

**#Clientes con más películas alquiladas que alguno de los que empiezan por ‘A’**

select first\_name,**count(rental\_id) total** from

customer join rental using (customer\_id)

group by first\_name

**having total> any (**select count(rental\_id) total from

customer join rental using (customer\_id)

where first\_name like 'a%'

group by first\_name

)

**ACCESO PROGRAMÁTICO**

Le das a server y luego a user and provilegies, add account y ahí escribes las cosas. Cuando se introduce la IP. Pones localhost en limit to hosts matching. Después cambias a la pestañita de adminstrative roles y eliges lo que puede hacer los distintos usuarios que se conectan a la database en función de su rol. Creada la base de datos le damos a aplicar y queda creada. Para conectarnos le damos en la pantalla de inicio del workbench le damos a la casita y introducimos el nombre de usuario y la IP, así como nos pedirá la contraseña.

**ESQUEMA ENTIDAD-RELACION**

Le das a **database🡪reverse engineering🡪seleccionas el usuario al que quieras conectarte🡪le das a next🡪seleccionas lo que quieres ver, en este caso solo tables🡪le das a next hasta que sala finish🡪Tienes el esquema E-R.**

**Camino azul 🡪 clave primaria**

**Camino verde 🡪foreign keys**

**INTRODUCCION**

Podemos distinguir 2 tipos de lenguajes, uno denominado lenguaje DML (SQL por ejemplo, que nos permite manejar los datos) y por otro lado DDL (lenguaje de definición de datos, que nos permite variar los datos de las tablas, modificarlas, eliminarlas…)

**HACER UNA CONEXIÓN**

Para ejecutar cualquier script primero es necesario ejecutar python, para ello nos vamos al **cmd** y ponemos **cd C:\Users\Usuario\AppData\Roaming\Microsoft\Windows\Start Menu**

Una vez estás en el directorio de python, escribes lo de python ./ruta…(siguiente párrafo).

Puedes usar el editor de texto que uses (notepad++), lo guardas como nombre\_file.py y después desde la consola de comandos de python pones: **python ./ruta/nombre\_file.py** y lo ejecutas.

**Import mysql.connector 🡪 Importamos la librería**

**Db = mysql.connector.connect(host=’localhost’, 🡪 Elhost, por lo general sera localhost**

**Port=”3306”, 🡪 El Puerto, por lo general 3306**

**User=”ejemplillo” ,🡪 El nombre de usuario del perfil**

**Passwd=”Fernando15”, 🡪 La contraseña del perfil**

**Db=”sakila”) 🡪 el nombre de la db**

**Db.close()**

**HACER UNA CONEXIÓN USANDO UN DICCIONARIO**

**Import mysql.connector.connect;**

**Config={**

**‘user’: ‘ejemplillo’, 🡪Nombre del usuario**

**‘password’:’Fernando15’, 🡪 Contraseña**

**‘host’:’localhost’, 🡪 IP del host (localhost)**

**‘database’: ‘sakila’, 🡪 Nombre de la base de datos**

**‘raise\_on\_warnings’: True, 🡪 Enseñar advertencias**

**}**

**Cnx=mysql.connector.connect(\*\*config); 🡪 Es FUNDAMENTAL el doble asterisco (\*\*);**

**Cnx.close();**

**CONEXION A DATABASE CON GESTION DE ERRORES**

**import mysql.connector;**

**from mysql.connector import errorcode;**

**config={**

**'user': 'ejemplillo',**

**'password':'Fernando15',**

**'host':'localhost',**

**'database': 'sakila',**

**'raise\_on\_warnings': True,**

**};**

**try:**

**cnx=mysql.connector.connect(\*\*config);**

**except mysql.connector.Error as err:**

**if err.errno == errorcode.ER\_ACCESS\_DENIED\_ERROR:**

**print ("Something is wrong with your username or password");**

**elif err.errno == errorcode.ER\_BAD\_DB\_ERROR:**

**print ("Database does not exist");**

**else:**

**print (err);**

**else:**

**print ("Everything Ok!");**

**cnx.close();**

**try, except, else 🡪 Van en bloque, hay que poner las 3.**

**QUERYING: CURSOR**

El cursor es un elemento que nos permitirá acceder a la base de datos una vez nos hemos conectado con ella. Tiene muchos parámetros, pero nosotros no los vamos a usar. Para usarlo primero es necesario conectarse a la base de datos, y posteriormente definimos el cursor con:

**cnx=mysql.connector.connect(…….)**

**cursor=cnx.cursor()**

Una vez hemos definido el cursor podemos lanzar las querys. Las querys serían como los comandos de SQL.

Podemos usar 2 métodos, pero el más cómodo es guardar la query en una variable y posteriormente usar esa variable con cursor.execute(variable):

**query=”select \* from sakila.actores”**

**cursor.execute(query)**

**QUERYING: COMANDO FETCHONE**

Cursor.fetchone() nos permite obtener la primera fila de una tabla, la cual hemos llamado previamente gracias a: query= “select \* from db.tabla”; cursor.execute(query). Por tanto sería:

**row=Cursor.fetchone()**

**print (row)**

PODEMOS COGER SOLO UNA COLUMNA, EL FETCHONE LO GUARDA COMO UNA LISTA:

**print(row[0]) 🡪 Nos devuelve el primer elemento, que lo normal es que sea el id**

**QUERYING: COMANDO FETCHALL()**

Para coger todas las filas de una tabla podemos optar por iterar el cursor.fetchone() o usar cursor.fetchall() de golpe (si queremos obtener unas filas determinadas usaremos el iterable, si queremos todas usaremos fetchall:

**all=cursor.fetchall()**

**for i in all:**

**print(i[1]) 🡪 Imprimimos la segunda columna, acuérdate que para llamar elementos de una lista tienes que iterar**

**QUERYING: COMANDO FETCHMANY()**

Cursor.fetchmany() es una especie de fetchall, pero especificas el número de filas que quieres que te devuelva, si te pasas de filas te devuelve todas y ya. Tiene un parámetro que es size, en el que especificas el número de filas que quieres, y al igual que en el fetchall() te devuelve una tupla en la que dentro tiene una lista por fila con los elementos de cada fila. Para usarlo:

**row=cursor.fetchmany(size=25)**

**for i in row:**

**print(row[1])**

**QUERYING: ITERAR SIN FETCH**

**query=”select \* from sakila”**

**cursor=db.cursor()**

**for i in cursor:**

**print(i[1])**

Podemos iterar directamente sobre el cursor para evitarnos el fetch. El ejemplo de arriba sería el equivalente a un cursor.fetchall()

**QUERYING: QUERYS PARAMÉTRICAS**

Se usa cuando no sabemos el id al que queremos aplicar la query, por ejemplo si queremos crear una query con una fila que tenga un determinado valor pero no conocemos su id, es decir, no podemos aplicar el where id=X, por eso en vez de poner un id podemos especificar una variable y cuando lo sepamos lo ponemos más tarde en el código sin que de ningún error. Para ello ponemos:

**query = “select \* from sakila.actor where id=%s and primer\_nombre=%s and …”**

**cursor.execute(query, (id,primer\_nombre,…,)) 🡪 IMPORTANTE LA COMA AL FINAL. El parámetro entre paréntesis es cómo quieres llamar tú a la variable, y puedes poner varios parámetros como por ejemplo:**

**cursor.execute(query, (id,nombre,apellido,)**

**EJEMPLO:**

**#EJEMPLO DE USO DE QUERYS PARAMÉTRICAS PARA IMPRIMIR LOS MATCH DE UN DETERMINADO ID INTRODUCIDO POR TECLADO**

**print ("------------------------------------------------------------")**

**import mysql.connector;**

**from mysql.connector import errorcode;**

**config={**

**'user': 'ejemplillo',**

**'password':'Fernando15',**

**'host':'localhost',**

**'database': 'sakila',**

**'raise\_on\_warnings': True,**

**};**

**db=mysql.connector.connect(\*\*config)**

**cursor=db.cursor()**

**try:**

**id=input("Introduce the ID of the actor: ")**

**query="SELECT \* FROM actor WHERE actor\_id=%s"**

**cursor.execute(query,(id,))**

**for result in cursor:**

**nombre=result[1]**

**apellido=result[2]**

**id=result[0]**

**update=result[3]**

**print ("El nombre del actor es: {0}".format(nombre))**

**print ("El apellido del actor es: {0}".format(apellido))**

**print ("El ID del actor es: {0}".format(id))**

**print ("La última actualizacion del actor es: {0}".format(update))**

**except ValueError:**

**print ("Something goes wrong")**

**finally:**

**db.close()**

**print ("Connection closed")**

**MODIFICACION DE LA BASE DE DATOS DESDE PYTHON**

Por defecto el autocommit está inactivo, por tanto es necesario indicar en el script cuándo queremos hacer el commit. Por ello, en el script será necesario poner db.commit() para que se apliquen los cambios, pero una vez hechos los cambios no se puede volver atrás, así que cuidado no la lies.

**Db.commit() 🡪 Se pone al final para aplicar de forma no reversible los cambios a la base de datos.**

También podemos activar el autocommit, qiue por defecto está desactivado, esto es más peligroso. Para ello escribimos:

**Db.autocommit=True**

**EJEMPLO USO DEL AUTOCOMMIT PARA MODIFICACION DE UNA BASE DE DATOS**

**import mysql.connector**

**from datetime import date, datetime**

**config={**

**'user': 'ejemplillo',**

**'password':'Fernando15',**

**'host':'localhost',**

**'database': 'sakila',**

**'raise\_on\_warnings': True,**

**};**

**db=mysql.connector.connect(\*\*config)**

**cursor=db.cursor()**

**db.autocommit=True #Aquí activamos el autocommit, todo cambio se aplicará instant**

**query="INSERT INTO actor (actor\_id, first\_name, last\_name,last\_update) VALUES (%s,%s,%s,%s)"**

**id=543**

**fname="FERNANDO"**

**lname="JIMENEZ"**

**update=datetime.now().date()**

**cursor.execute(query,(id,fname,lname,update,))**

**querycheck="SELECT \* FROM actor"**

**cursor.execute(querycheck)**

**for row in cursor:**

**id=row[0]**

**fname=row[1]**

**lname=row[2]**

**update=str(row[3])**

**print ("ID={0}, first\_name={1}, last\_name={2}, last\_update={3}".format(id,fname,lname,update))**

**db.close()**

**INTRODUCIR VARIAS FILAS A LA VEZ EN UNA LISTA YA CREADA (TUPLA EN LA QUE CADA ELEMENTO ES UNA LISTA)**

Si queremos introducir varias filas de datos en una tabla, debemos de crear una tupla en la que cada elemento sea una lista cuyos elementos son las columnas de la fila, es decir, los datos de cada fila. Una vez creada esta tupla compleja cuando hagamos el execute del query simplemente pondremos después de especificar el query el nombre de la lista, no hace falta poner (id, fname, lname,) dado que la lista ya está creada. Por ello, sería:

**Nombre\_tupla=[(‘PEPE’,’GUTIERREZ’,7,date(12,3,19)),(‘JOSE’,’LOPEZ’,4,date(3,5,18))]**

**Query=”INSERT INTO tabla (nombre, apellido,nota,fecha) VALUES(%s,%s,%s,%s)”**

**Cursor.executemany(query,nombre\_tupla)**

**IMPORTANTE: NO ES cursor.execute, es cursor.executeMANY**

**CONTAR LOS ELEMENTOS AÑADIDOS O DE LA TABLA AL COMPLETO**

Para contar los elementos que hemos añadido o los que tiene la nueva tabla, debemos hacer un fetchall() seguido de un print(cursor.rowcount). Por ello:

**cursor.execute(query)** #Ejecutamos la query

**all=cursor.fetchall()** #Hacemos un fetchall() para poder contar los elementos

**print (cursor.rowcount)** #Imprimimos el count

**E-R y paso a tablas**

El es quema éste de cuadrado y rombos es el de entidad relación, y lo tienes que pasar a un conjunto de tablas.

**Entidades**

* **Entidades fuertes** 🡪 Son los atributos de toda la vida, los atributos simples que formarán las columnas de nuestra tabla. Por ejemplo si tenemos la tabla **coche,** entidades fuertes serán número de bastidor (que sería la primary key), kilómetros, año de fabricación…
* **Atributos compuestos 🡪** Guardamos la parte que lo componen, pero no el atributo compuesto en sí.
* **Atributos multivalorados** 🡪 En este caso será necesario crear una segunda tabla que introduzca esta nueva información y que pueda relacionarse con la tabla inicial, por lo que será necesario que exista una relación entre ambas, que se consigue introduciendo un mismo id en ambas tablas. Por ejemplo, si tenemos una tabla denominada **coches** con distintos atributos (id\_coche es primary key), y uno de ellos es el color, puede haber varios colores para un coche, por lo que dicho atributo es multivalorado y por tanto crearemos una segunda tabla que se denomine **colores** y que tenga como atributos id\_coche (primary key) y el color.
* **Atributos derivados 🡪** Son atributos que se calculan a partir de atributos que ya tenemos, podemos incluirlos como un atributo más o no almacenar las columnas que provengan de atributos derivados. Por ejemplo, en la tabla **coche** si tenemos un atributo que es **antigüedad** (que hace falta calcularlo todo el rato porque va variando) podemos o no incluirlo o calcularlo todo el rato.
* **Entidades débiles 🡪** Su identificador estará definido por el identificador de la entidad fuerte + el identificador de la entidad débil.

**TIPOS DE RELACIONES**

* **Relación N:M 🡪** 1 de X puede ser n de Y, pero a su vez 1 de Y puede ser m de X, es decir, las entidades siguen una relación de muchos a muchos. Ésta relación genera un esquema en el que no solo las entidades son tablas, sino que la propia relación genera una nueva tabla que relaciona los id de los 2 elementos (así como puede incluir otros atributos adicionales. Por ejemplo, podemos querer guardar qué videojuegos juega cada alumno, cuántas horas ha echado a cada uno de ellos y su puntuación máxima. Por tanto, el esquema E-R quedaría:

nombre

Id\_videojuego

videojuego

horas

puntuacion

nombre

ID\_alumno

juega

alumno

Por tanto tendremos las siguientes tablas:

ALUMNO 🡪 ID\_alumno (primary key), nombre

JUEGA 🡪 ID\_alumno (references alumno), id\_videojuego (references videojuego), horas, puntuación, primary ke (id\_alumno, id\_videojuego)

VIDEOJUEGO 🡪 id\_videojuego (primary key), nombre

* **Relación 1:N 🡪** En este caso, si tenemos una relación 1:N o N:1 tendremos 2 entidades, en el que el elemento 1 tendrá un id y en el elemento N se incluirá tanto su propio id como el id del elemento 1, es decir, la foreign key de la entidad N será la primary key de la entidad 1. Por ejemplo, podemos querer saber qué despacho corresponde a cada profesor, así como características tanto del despacho como de cada profesor. Por tanto, el esquema E-R quedaría:

situacion

dni

nombre

descripcion

metros

tiene

ID despacho

despachos

ffffff

Profesores

PROFESORES: dni (primary key), nombre, ID\_despacho

DESPACHOS: ID\_despacho (primary key) , descripción, situación, metros

**CREAR UNA CLAVE FORANEA**

Para crear una clave foránea, primero es necesario establecer una primary key en la otra tabla con la que la queremos relacionar. Es decir, tendremos una primary key y una foreign key. Para establecer una foreign key es necesario la siguiente estructura:

**CONSTRAINT nombre\_que\_quieras\_dar\_a\_la\_FK FOREIGN KEY (columna\_que\_será\_la\_FK) REFERENCES tabla\_con\_la\_que\_se\_relaciona (columna\_de\_la\_otra\_tabla\_que\_funciona\_como\_primary\_key)**

Por ejemplo, si tenemos una base de datos que se compone como en el ejemplo anterior de alumnos quej uegan a videojuegos de 3 tablas, que siguen una relación N:M, por lo que necesitaremos 3 tablas, una de ellas albergará 2 FK, por lo que el código quedaría:

**create table alumno (**

**ID\_alumno int not null auto\_increment primary key,**

**nombre varchar(30));**

**create table videojuego (**

**ID\_videojuego int auto\_increment not null primary key,**

**nombre varchar(30));**

**create table juega (**

**ID\_juega int auto\_increment not null primary key,**

**ID\_alumno\_juega int,**

**ID\_videojuego\_juega int,**

**horas float(4,2) not null,**

**puntuacion int(30),**

**constraint FK\_alumno foreign key (ID\_alumno\_juega) references alumno (ID\_alumno),**

**constraint FK\_videojuego foreign key (ID\_videojuego\_juega) references videojuego (ID\_videojuego))**