EXERCISES QUE TRABAJAREMOS EN EL CUE

- **EXERCISE 1: PAQUETES DE INSTALACIÓN PARA BASES DE DATOS**
- **EXERCISE 2: MIGRACIONES EN DJANGO**
- **EXERCISE 3: DJANGO ORM AND QUERYSET**

EXERCISE 1: PAQUETES DE INSTALACIÓN PARA BASES DE DATOS

POSTGRESQL

Django es compatible con PostgreSQL 11, y superior. Se requiere psycopg2 2.8.4 o superior, aunque se recomienda la última versión.

`Psycopg` es el adaptador PostgreSQL más popular para el lenguaje de programación Python. Su núcleo es una implementación completa de las especificaciones Python DB API 2.0. Varias extensiones permiten el acceso a muchas de las funciones que ofrece PostgreSQL.

INSTALACIÓN DE PSYCOPG2

Pre - requisitos:

- Python versión de 3.6 a 3.10.
- PostgreSQL server versión de 7.4 a 14.
- PostgreSQL client library versión 9.1.

Primero instale los requisitos previos (no son necesarios en Windows, adáptese a su distribución Linux):

```
sudo apt install python3-dev libpq-dev
```

Luego, instale el módulo:

```
pip instalar psycopg2
```

Para más información de las dependencias de instalación, puedes ver este enlace.

MARIADB

Django es compatible con MariaDB 10.3, y superior. Para usarlo, aplique el backend de MySQL, que se comparte entre los dos. Consulte las notas de MySQL para obtener más detalles.

MYSQL

Django es compatible con MySQL 5.7, y superior. La función inspectdb de Django utiliza la base de datos information_schema, que contiene datos detallados sobre todos los esquemas de la base de datos.

Django espera que la base de datos admita Unicode (codificación UTF-8), y le delega la tarea de hacer cumplir las transacciones y la integridad referencial. Es importante tener en cuenta el hecho de que MySQL no aplica los dos últimos cuando se usa el motor de almacenamiento MyISAM.

MySQL tiene un par de controladores que implementa la API de base de datos de Python descrita en PEP 249: mysqlclient es un controlador nativo. Es la opción recomendada.

MySQL Connector/Python es un controlador de Python puro de Oracle, el cual no requiere la biblioteca cliente de MySQL, ni ningún módulo de Python fuera de la biblioteca estándar. Estos controladores son seguros para subprocesos y proporcionan agrupación de conexiones.

Además de un controlador API de base de datos, Django necesita un adaptador para acceder a los controladores de base de datos desde su ORM. Éste proporciona un adaptador para **mysqlclient,** mientras que **MySQL Connector/Python** incluye el suyo propio.

PROCESO DE INSTALACIÓN

Pre - requisito: Tener instalado la versión de MySQL Community Edition (CE).

WINDOWS

Construir mysqlclient en Windows es muy difícil. Pero hay algunos paquetes binarios que puedes instalar fácilmente, y luego haciendo uso de un ambiente virtual.

Para compilar desde la fuente, descarga MariaDB C Connector e instálalo. Una vez que tenga instalado el conector y una versión adecuada de Visual Studio para su versión de Python:

```
$ pip install mysqlclient
```

LINUX DEBIAN/UBUNTU:

Es posible que deba instalar los encabezados y bibliotecas de desarrollo de Python 3 y MySQL de la siguiente manera:

```
$sudo apt-get install python3-dev default-libmysqlclient-dev build-
essential
```

Seguidamente:

```
$ pip install mysqlclient
```

Referencia aquí.

SQLITE

Django es compatible con SQLite 3.9.0, y versiones posteriores.

SQLite proporciona una excelente alternativa de desarrollo para aplicaciones que son predominantemente de solo lectura, o que requieren una instalación más corta. Sin embargo, al igual que con todos los servidores de bases de datos, existen algunas diferencias específicas de SQLite que debe tener en cuenta.

Django está configurado para usar SQLite por defecto cuando comienzas tu proyecto de sitio web usando las herramientas estándar (django-admin). Es una gran elección cuando estás empezando, pues no requiere configuración o puesta en marcha adicional.

ORACLE

Django es compatible con las versiones 19c, y posteriores, de Oracle Database Server. Se requiere la versión 7.0 o superior del controlador `cx_Oracle` Python. `cx_Oracle` es una biblioteca de Python que podemos usar para conectarnos a esquemas de Oracle. La última versión de `cx_Oracle` admite python3.6, y versiones posteriores.

Admite la conexión a la base de datos Oracle 9.2, 10, 11, 12, 18, 19 y 21. `cx_Oracle` expone funciones de Python para ejecutar cada tarea. Los usuarios solo necesitan instalar el paquete, y llamar a las funciones requeridas. `cx_Oracle` proporciona funcionalidades principales para realizar manipulaciones de datos en la base de datos Oracle mediante consultas y procedimientos almacenados.

Instalación:

```
$ pip install cx_Oracle
```

EXERCISE 2: MIGRACIONES EN DJANGO

Para convertir un modelo en una estructura de tabla de bases de datos en Django se requiere más que la estructura, se requiere la migración; que es pasar del modelo de datos a las tablas de la base de datos. Este es un tipo especial de archivo Python que contiene las instrucciones que necesita Django para crear tablas de base de datos en su nombre.

Django usa el modelo de datos para ingresar estas instrucciones `migrate`, y ejecutar los comandos para aplicar estos cambios a la base de datos.

Los pasos importantes para realizar las migraciones son:

- Cree un archivo de migración con instrucciones para modificar las tablas de la base de datos.
- Ejecute el código contenido en el archivo de migración para migrar las tablas de la base de datos.

Este proceso de dos pasos garantiza que solo se apliquen explícitamente los cambios que desea realizar. Si el archivo de migración contiene errores, o le faltan campos importantes, puede solucionar el problema antes de aplicar los cambios a la base de datos.

Este proceso hace que sea muy fácil interactuar con un sólido sistema de administración de bases de datos, incluso para los desarrolladores web menos experimentados. No es necesario que modifique la base de datos mediante consultas SQL complejas o la consola de administración basada en navegador.

Todo lo que tiene que hacer es definir su modelo de datos, migrar sus cambios, y estará listo para comenzar. La migración es un gran ejemplo de la función de creación rápida de prototipos de Django.

Django actúa como un desarrollo incremental para ayudar a arrancar el desarrollo web, por lo que algunos modelos de datos internos están pre - empaquetados, y la base de datos SQLite se crea automáticamente cuando aplica la primera migración.

Para migrar el modelo de datos interno de Django, y crear una base de datos inicial, se usa el siguiente comando:

```
$ python3 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

EXERCISE 3: DJANGO ORM AND QUERYSET

Un conjunto de consultas (`queryset`) son simplemente una lista de objetos de los modelos de Django. Cuando se usa el ORM de Django, se crea una nueva fila en la tabla, que es simplemente como crear

un nuevo objeto de la clase Model. Django ORM asigna estos objetos de modelo a una consulta de base de datos relacional.

Cualquier consulta SQL se puede escribir fácilmente como un conjunto de consultas de Django. Puede probar por separado la consulta para Crear, Filtrar, Actualizar, Ordenar, entre otros, en Django Shell o en `views.py`.

La sintaxis para un conjunto de consultas es siempre como:

```
ModelName.objects.method_name(arguments)
```

SQL QUERY VS DJANGO ORM QUERYSET

SQL QUERY	DJANGO ORM QUERYSET
`select * from table;`	`Model.objects.all()`
`select * from table where fieldname=value;`	`Model.objects.filter(fieldname=value)`
`select distinct column_names from table;`	`Model.objects.distinct(column_names)`
`insert into table(field1, field2, …) values (value1, value2, …);`	`Model.objects.create(field1=value1, field2=value2, …)`
`update table set column_name = value where condition;`	`Model.objects.filter(condition).update( column_name=value)`
`delete from table where condition`	`Model.objects.filter(condition).delete( )`

`select columns from table order_by column asc|desc;`	`Model.objects.filter(condition).order_by (column_for_asc, -column_for_desc)`
`select columns from table where conditon1 and condition2`	`Model.objects.filter(condition1, condition2)`
`select columns from table where conditon1 or condition2`	`Model.objects.filter(condition1) | Model.objects.filter(condition2)`
`select columns from table where not condition`	`Model.obejcts.exclude(condition)`
`select max(column) from table;`	`Model.obejects.filter(column).max()`
`select min(column) from table;`	`Model.obejects.filter(column).min()`
`select avg(column) from table;`	`Model.obejects.filter(column).avg()`
`select sum(column) from table;`	`Model.obejects.filter(column).sum()`
`select count(column) from table;`	`Model.obejects.filter(column).count()`