PAQUETES DE INSTALACIÓN DE BASES DE DATOS

TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE:

- Paquetes de Instalación
- Django como ORM
- Migraciones en Django
- Sintaxis de consultas en ORM

PAQUETES DE INSTALACIÓN

Por defecto, Django está configurado con la base de datos sqlite3, por lo que crea automáticamente una base de datos para el proyecto. Django puede configurarse con los gestores de bases de datos: PostrgreSQL, Oracle, y Mysql. Para configurar el acceso a la base de datos editamos el archivo del directorio del proyecto `setting.py`.

Los paquetes a ser instalados son:

- **Database Python package**: `pip installation syntax`. PIP es un administrador de paquetes para paquetes, o módulos de Python.
- **PostgreSQL psycopg*X*:** `pip install psycopgX`. ***X*** se refiere a la versión, actualmente se encuentra disponible 1, 2 o 3, el cual dependerá de las versiones que se estén trabajando en el proyecto, y su compatibilidad.
- **MySQL mysql-python:** `pip install mysql-python`
- **Oracle cx Oracle:** `pip install cx Oracle`

DJANGO COMO ORM

El ORM de Django es una implementación del concepto de mapeo de objeto relacional (ORM). Una de las características más poderosas de Django es su Mapeador Relacional de Objetos (ORM), que le permite interactuar con su base de datos, como lo haría con instrucciones SQL (Structured Query Language).

El marco web de Django incluye esa capa de mapeo relacional de objetos (ORM) predeterminada, la cual se puede usar para interactuar con los datos de la aplicación, es una biblioteca de código que automatiza

la transferencia de datos almacenados en tablas de bases de datos relacionales a objetos que se usan más comúnmente en el código de la aplicación.

DJANGO COMO ORM

Las migraciones en Django se encargan de que toda la lógica asociada con la base de datos sea reflejada en la base de datos como tal. Cuando se inicia un proyecto en Django, hay una serie de migraciones necesarias que Django utiliza para mantener información de las sesiones y administradores. Esta siempre será la primera migración que realizaremos.

```
$ python manage. py migrate
```

Las migraciones son la forma en que Django propaga los cambios que realiza en sus modelos (agregando un campo, eliminando un modelo, entre otros) en el esquema de su base de datos. Están diseñados para ser en su mayoría automáticos, pero necesitará saber cuándo realizar migraciones, cuándo ejecutarlas, y los problemas comunes con los que podría encontrarse.

Hay varios comandos que se usarán para interactuar con las migraciones y el manejo de Django del esquema de la base de datos:

- `Migrate` - Es responsable de aplicar y no aplicar las migraciones.
- `makemigrations` - Es responsable de crear nuevas migraciones basadas en los cambios que ha realizado en sus modelos. Empaqueta los cambios de su modelo en archivos de migración individuales, análogos a los commit.
- `Sqlmigrate` - Muestra las instrucciones SQL para una migración.
- `Showmigrations` - Enumera las migraciones de un proyecto y su estado.

Los archivos de migración para cada aplicación viven en un directorio de "migraciones" dentro de esa aplicación, y están diseñados para distribuirse como parte de su código base. Debería hacerlos una vez en su máquina de desarrollo, y luego ejecutar las mismas migraciones en las máquinas de producción.

Es posible anular el nombre del paquete que contiene las migraciones por aplicación modificando la configuración `MIGRATION_MODULES`.

SINTAXIS DE CONSULTAS EN ORM

Las consultas en ORM permite automáticamente que se escriban consultas SQL correctas y optimizadas, eliminando así la molestia para los desarrolladores. Hacen que el código sea más fácil de actualizar, mantener y reutilizar, ya que el desarrollador puede pensar en los datos y manipularlos como objetos.

El ORM de Django nos proporciona dos métodos para filtrar los `querysets`: `filter` y `exclude`. También podemos acceder a un único registro utilizando el método `get`.

El método `filter` se utiliza para filtrar un conjunto de datos, y devuelve un `Queryset` que cumple con las condiciones indicadas. El método permite que se le pasen varios parámetros para concatenar filtros. Por ejemplo, recuperar todos los usuarios de un país:

```
1 Person.objects.filter(country_iso="CANADA")
```

El método `exclude` tiene un comportamiento muy similar al método `filter`. También devuelve un `Queryset` filtrado, pero en este caso excluyendo los registros que cumplen las condiciones indicadas por los parámetros. Por ejemplo, devolver todos los usuarios que no son de un país concreto:

```
1 Person.objects.exclude(country_iso="MEXICO")
```

Se puede utilizar estos métodos en una misma sentencia, por ejemplo, al recuperar todos los usuarios que se dieron de alta un día y proporcionaron su email.

```
1 Person.objects.filter(created_at=datetime(2022,7,18)).exclude(email="")
```

El método `get` devuelve un objeto que cumpla las condiciones indicadas.

```
1 Person.objects.get(passport="12345678A")
```

Si los filtros añadidos no encuentran ningún resultado, se mostrará una excepción de tipo `Model.DoesNotExist`; y si se encuentran múltiples resultados, la excepción será `Model.MultipleObjectsReturned`.

Se puede tener en cuenta que el paginado de una aplicación web al final se convierte en un filtro en las consultas a la base de datos para controlar el número de registros devueltos en ellas.