TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE:

- Uso de Migraciones
- Implementando Migraciones en Django
- Creando migraciones con makemigrations
- Aplicando migraciones nuevas y migraciones existentes

¿QUÉ ES UNA MIGRACIÓN?

Es el proceso mediante el cual se realiza una transferencia de datos que puede ser de unos sistemas de almacenamiento de datos a otros, de unos formatos de datos a otros, o entre diferentes sistemas informáticos. Las migraciones en Django propagan 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.

PROBLEMA QUE RESUELVE

La migración es un poderoso proceso mediante el cual realizamos una transferencia de datos con seguridad para no perder información. Se pueden considerar como un sistema de control de versiones para su esquema de base de datos.

Los archivos de migración para cada aplicación viven en un directorio de "migraciones" dentro de esa aplicación, y están diseñados para comprometerse y distribuirse como parte de su código base.

UTILIZANDO LAS MIGRACIONES DE DJANGO

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.

- `sqlmigrate`, muestra las instrucciones SQL para una migración.
- `showmigrations`, enumera las migraciones de un proyecto y su estado.

Las migraciones se ejecutarán de la misma manera, en el mismo conjunto de datos, y producirán resultados consistentes, lo que significa que, lo que se ve en el desarrollo y la puesta en escena es, en las mismas circunstancias, exactamente lo que sucederá en la producción.

Django realizará migraciones para cualquier cambio en sus modelos o campos, incluso opciones que no afecten a la base de datos, ya que la única forma en que puede reconstruir un campo correctamente es tener todos los cambios en el historial, y es posible que necesite dichas opciones en algunas migraciones de datos más adelante (por ejemplo, si ha configurado validadores personalizados).

PARÁMETROS

comando: `django-admin`

`makemigrations <my_app>:` Generar migraciones para `my_app`.

`makemigrations:` Generar migraciones para todas las aplicaciones.

`makemigrations --merge:` Resolver conflictos de migración para todas las aplicaciones.

`makemigrations --merge <my_app>:` Resolver conflictos de migración para `my_app`.

`makemigrations --name <migration_name> <my_app>:` Genera una migración para `my_app` con el nombre `migration_name`.

`migrate <my_app>:` Aplicar migraciones pendientes de `my_app` a la base de datos.

`migrate:` Aplicar todas las migraciones pendientes a la base de datos.

`migrate <my_app> <migration_name>:` Aplicar o no aplicar hasta el nombre de `migration_name`.

`migrate <my_app> zero:` Desplegar todas las migraciones en `my_app`.

`sqlmigrate <my_app> <migration_name>:` Imprime el SQL para la migración nombrada.

`showmigrations:` Muestra todas las migraciones para todas las aplicaciones.

`showmigrations <my_app>:` Muestra todas las migraciones en `my_app`.

CREANDO MIGRACIONES CON MAKEMIGRATIONS

El `makemigrations` es responsable de empaquetar los cambios de su modelo en archivos de migración individuales, análogos a los `commits`; y `migrate` es responsable de aplicarlos a su base de datos.

Los tres pasos para trabajar las migraciones son:

- Cambiar los modelos (en models.py).
- Ejecutar el comando Python `manage.py makemigrations`, para crear migraciones para esos cambios.
- Ejecutar el comando Python `manage.py migrate`, para aplicar esos cambios a la base de datos.

```
./manage.py makemigrations polls
```

Mediante la ejecución de `makemigrations`, le estás indicando a Django que has hecho algunos cambios a sus modelos, y que deseas que los cambios se guarden como la migración.

Las migraciones son la forma en que Django almacena cambios a sus modelos (y por tanto el esquema de base de datos), es decir, son simplemente los archivos en el disco. Puedes leer la migración para su nuevo modelo si se quiere; que es el archivo: `polls/migrations/0001_initial.py`.

Para ver el SQL que generará cuando cree la `db`, use `sqlmigrate`.

Por ejemplo, para ver `0001_initial.py` `0002_auto_20140808_1242.py`, solo poner el primer número `0001` o `0002`.

Esto no genera la base de datos, solo muestra el SQL que generará cuando se haga `migrate`

```
./manage.py sqlmigrate polls 0001
```

COMPROBAR EL MODELO

El siguiente comando solo comprueba si hay algún error en el modelo, pero no ejecuta nada.

```
./manage.py check
```

Migrar la base de datos.

```
./manage.py migrate
```

El comando `migrate` toma todas las migraciones que no han sido aplicadas, y las actualiza en su base de datos en esencia, la sincronización de los cambios realizados en sus modelos con el esquema en la base de datos.

Las migraciones son muy potentes, y permiten cambiar tus modelos con el tiempo a medida que se desarrolle el proyecto, sin la necesidad de eliminar la base de datos o las tablas y hacer otros nuevos.

APLICANDO NUEVAS MIGRACIONES

Las migraciones iniciales para una aplicación son aquellas que crean la primera versión de las tablas de esa aplicación. Por lo general, una aplicación tendrá una migración inicial, pero en algunos casos de interdependencias complejas del modelo, puede tener dos o más.

Éstas se marcan con un atributo de clase `initial = True` en la clase de migración. Si no se encuentra un atributo de clase `initial`, una migración se considerará "inicial" si es la primera migración en la aplicación (es decir, si no tiene dependencias de ninguna otra migración en la misma aplicación).

Cuando se usa la opción `migrate --fake-initial`, estas migraciones iniciales se tratan especialmente. Para una migración inicial que crea una o más tablas (operación `CreateModel`), Django comprueba que todas esas tablas ya existen en la base de datos, y la aplica. De manera similar, para una migración inicial que agrega uno o más campos (operación `AddField`), Django verifica que todas las columnas respectivas ya existan en la base de datos, y la aplica. Sin `--fake-initial`, las migraciones iniciales no se tratan de manera diferente a cualquier otra.

APLICANDO MIGRACIONES EXISTENTES

Las migraciones se almacenan como un formato en el disco, denominado aquí "archivos de migración". Estos archivos son en realidad archivos normales de Python, con un diseño de objeto acordado, escrito en un estilo declarativo.

Un archivo básico de migración se ve así:

```
from django.db import migrations, models
class Migration(migrations.Migration):
    dependencies = [('migrations', '0001_initial')]
    operations = [
        migrations.DeleteModel('Tribble'),
```

```
        migrations.AddField('Author', 'rating',
        models.IntegerField(default=0)),
    ]
```

Lo que Django busca cuando carga un archivo de migración (como un módulo de Python), es una subclase de django.`db.migrations.Migration` llamada `Migration`. Luego, inspecciona este objeto en busca de cuatro atributos, de los cuales solo dos se usan la mayor parte del tiempo:

- `Dependencies:` una lista de migraciones de la que depende.
- `Operations:` una lista de clases de Operation que definen lo que hace esta migración.

Las operaciones son un conjunto de instrucciones declarativas que le indican a Django qué cambios de esquema hay que hacer. Django las escanea, y construye una representación en memoria de todos los cambios de esquema en todas las aplicaciones, y las utiliza para generar el SQL que realiza los cambios de esquema.

Esa estructura en memoria también se usa para determinar cuáles son las diferencias entre sus modelos, y el estado actual de sus migraciones; Django ejecuta todos los cambios, en orden, en un conjunto de modelos en memoria para obtener el estado de sus modelos la última vez que ejecutó `makemigrations`. Luego, los usa para compararlos con los de sus archivos `models.py`, y así calcular lo que ha cambiado.