1. Example for custom migration(schema_editor):

```
      from django.db import migrations, models

      def create_temporary_table(apps, schema_editor):
            # Get the model class
            Person = apps.get_model('your_app_name', 'Person')

            # Access the SchemaEditor to create a temporary table
            schema_editor.execute(
                  "CREATE TEMPORARY TABLE temp_person_data AS SELECT id,
first_name, last_name FROM your_app_name_person"
            )

      def reverse_create_temporary_table(apps, schema_editor):
            schema_editor.execute("DROP TABLE temp_person_data")

      class Migration(migrations.Migration):

            dependencies = [
                  ('your_app_name', 'previous_migration'),
            ]

            operations = [
                  migrations.RunPython(create_temporary_table,
reverse_create_temporary_table),
            ]
```

#1.

```
      class Shoe(models.Model):
            brand = models.CharField(
                  max_length=25,
            )
            size = models.PositiveIntegerField()

            python manage.py sqlmigrate main_app 0001_initial
```

#2.

```
      python manage.py makemigration main_app --name migrate_unique_brands --empty

      class UniqueBrands(models.Model):
            brand_name = models.CharField(
                  max_length = 25,
                  unique = True,
            )

      def create_unique_brands(apps, schema_editor):
            shoe = apps.get_model('main_app', 'Shoe')
            unique_brands = apps.get_model('main_app', 'UniqueBrands')
            unique_brand_names = shoe.objects.values_list('brand',
flat=True).distinct()
            # Below is option 1 - better to use option 2
            # for brand_name in unique_brand_names:
            #     unique_brands.create(brand_name=brand_name)
            unique_brands_to_create = [unique_brands(brand_name=brand_name) for
brand_name in unique_brand_names]
            unique_brands.objects.bulk_create(unique_brands_to_create)
      def delete_unique_brands(apps, schema_editor):
            unique_brands = apps.get_model('main_app', 'UniqueBrands')
            unique_brands.objects.all().delete()


      class Migration(migrations.Migration):
            dependencies = [
                  ('my_app', '0002_uniquebrands'),
            ]

            operations = [
                  migrations.RunPython(create_unique_brands,
reverse_code=delete_unique_brands()


      print(Shoe.objects.values_list('brand', flat=True).distinct())
      #flat=True ->(converts from tuple to str)

#3.
      class EventRegistration(models.Model):
            event_name = models.CharField(
                  max_length=60,
            )
            participant_name = models.CharField(
                  max_length=50,
            )
            registration_date = models.DateField()

            def __str__(self):
                  return f'{self.participant_name} - {self.event_name}'


      @admin.register(EventRegistration)
      class EventRegistrationAdmin(admin.ModelAdmin):
            list_display = ['event_name', 'participant_name', 'registration_date']
            list_filter = ['event_name', 'registration_date']
            search_fields = ['event_name', 'participant_name']

#4.
      class Student(models.Model):
            first_name = models.CharField(
                  max_length=50,
            )
            last_name = models.CharField(
                  max_length=50,
            )
            age = models.PositiveIntegerField()
            grade = models.CharField(
                  max_length=10,
            )
            date_of_birth = models.DateField()
            def __str__(self):
                  return f'{self.first_name} {self.last_name}'


      @admin.register(Student)
      class StudentAdmin(admin.ModelAdmin):
            list_display = ['first_name', 'last_name', 'age', 'grade']
            list_filter = ['age', 'grade', 'date_of_birth']
            search_fields = ['first_name']
            fieldsets = (
                  ('Personal Information', {
                        'fields': ('first_name',
                                   'last_name',
                                   'age',
                                   'date_of_birth',
                                   )
                  }),
                  ('Academic Information', {
                        'fields': ('grade',),
                        # 'classes': ['collapse'],
                  }),
            )

#5.
      class Supplier(models.Model):
            name = models.CharField(
                  max_length=100,
            )
            contact_person = models.CharField(
                  max_length=50,
            )
            email = models.EmailField(
                  unique=True,
            )
            phone = models.CharField(
                  max_length=20,
                  unique=True,
            )
            address = models.TextField()

            def __str__(self):
                  return f'{self.name} - {self.phone}'

      @admin.register(Supplier)
      class SupplierAdmin(admin.ModelAdmin):
            list_display = ['name', 'email', 'phone']
            list_filter = ['name', 'phone']
            list_per_page = 20
            readonly_fields = ('email',)
            search_fields = ['email', 'contact_person', 'phone']
            fieldsets = (
                  ('Information', {
                        'fields':(
                              'name',
                              'contact_person',
                              'email',
                              'address',
                        )}),
            )

#6.
            class Person(models.Model):
                  name = models.CharField(
                        max_length=40,
                  )
                  age = models.PositiveIntegerField()
                  age_group = models.CharField(
                        max_length=20,
                        default='No age group',
                  )
                  def __str__(self):
                        return  f'Name: {self.name}'


            def set_age_group(apps, schema_editor):
                  person = apps.get_model('main_app', 'Person')
                  persons = person.objects.all()
                  for person_record in persons:
                        if  person_record.age <= 12:
                              person_record.age_group = 'Child'
                        elif 13 <= person_record.age <= 17:
                              person.age_group = 'Teen'
                        elif person_record.age >= 18:
                              person_record.age_group = 'Adult'

                  person.objects.bulk_update(persons, ['age_group'])


            def set_age_group_defautl(apps, schema_editor):
                  person = apps.get_model('main_app', 'Person')
                  age_group_default = person._meta.get_field('age_group').default
#retuns characteristics of the column itself,but not the data inside
                  for p in person.objects.all():
                        p.age_group = age_group_default
                        p.save()

            # makemigrations main_app --name migrate_age_group --empty
            class Migration(migrations.Migration):
                  dependencies = [
                        ('my_app', '0007_person'),
                  ]

                  operations = [
                        migrations.RunPython(set_age_group,
reverse_code=set_age_group_defautl)
                  ]

#7.
      class Smartphone(models.Model):
            brand = models.CharField(
                  max_length=100,
            )
            price = models.DecimalField(
                  max_digits=10,
                  decimal_places=2,
                  default=1,
            )
            category = models.CharField(
                  max_length=20,
                  default='empty',
            )

      def set_price(apps, schema_editor):
            MULTIPLY_PRICE = 120
            smartphone_model = apps.get_model('main_app', 'Smartphone')
            for smartphone in smartphone_model.objects.all():
                  smartphone.price = len(smartphone.brand) * MULTIPLY_PRICE
                  smartphone.save()

      def set_category(apps, schema_editor):
            smartphone_model = apps.get_model('main_app', 'Smartphone')
            for smartphone in smartphone_model.objects.all():
                  if smartphone.price >= 750:
                        smartphone.category = 'Expensive'
                  else:
                        smartphone.category = 'Cheap'

                  smartphone.save()

      def set_category_and_price(apps, schema_editor):
            set_price(apps, schema_editor)
            set_category(apps, schema_editor)

      def reverse_code(apps, schema_editor):
            smartphone_model = apps.get_model('main_app', 'Smartphone')
            for smartphone in smartphone_model.objects.all():
                  smartphone.price = 0
                  smartphone.category = 'empty'
                  smartphone.save()

      class Migration(migrations.Migration):
            dependencies = [
                  ('my_app', '0008_smartphone'),
            ]

            operations = [
                  migrations.RunPython(set_category_and_price,
reverse_code=reverse_code),
            ]

#8.

from django.utils import  timezone

class Order(models.Model):
    STATUS_CHOICES = (
        ('P', 'Pending'),
        ('Com', 'Completed'),
        ('Can', 'Cancelled'),
    )
    product_name = models.CharField(
        max_length= 30,
    )
    customer_name = models.CharField(
        max_length= 100,
    )
    order_date = models.DateField()
    status = models.CharField(
        max_length=30,
        choices=STATUS_CHOICES,
    )
    amount = models.PositiveIntegerField(
        default=1,
    )
    product_price = models.DecimalField(
        max_digits=10,
        decimal_places=2,
    )
    total_price = models.DecimalField(
        max_digits=10,
        decimal_places=2,
        default=0,
    )
    warranty = models.CharField(
        default='No warranty',
    )
    delivered = models.DateField(
        null=True,
        blank=True,
    )

    def __str__(self):
        return f'Order #{self.pk} - {self.customer_name}'

def update_delivery_warranty(apps, schema_editor):
    order_model = apps.get_model('main_app', 'Order')
    for order in order_model.objects.all():
        if order.status == 'Pending':
            order.delivered = order.order_date + timezone.timedelta(days=3)
            order.save()
        elif order.status == 'Completed':
            order.warranty = '24 months'
            order.save()
        elif order.status == 'Cancelled':
            order.delete()

def reverse_delivery_warranty(apps, schema_edior):
    order_model = apps.get_model('main_app', 'Order')
    for order in order_model.objects.all():
        if order.status == 'Pending':
            order.delivered = None
        elif order.status == 'Completed':
            order.warranty = 'No warranty'

        order.save()


class Migration(migrations.Migration):
    dependencies = [
        ('main_app', '0011_order'),
    ]
    operations = [
        migrations.RunPython(update_delivery_warranty, reverse_delivery_warranty())
    ]
```