НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ “КПІ”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра спеціалізованих комп'ютерних систем

Лабораторна робота №2

*з дисципліни*

*“Організація баз даних”*

Виконав студент гр. КВ-22

Ткаченко Роман

Перевірив

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ 2014

**Завдання**: 17 варіант

Вид тригера: для кожного рядка;

Часовi параметри події: щотижня;

Рівні ізоляції транзакцій: **SERIALIZABLE**, **REPEATABLE** **READ**;

**Список класів Django ORM:**

class Produser(models.Model):

id = models.IntegerField(primary\_key=True)

name = models.CharField(max\_length=45, blank=True)

rating = models.IntegerField(blank=True, null=True)

guitar = models.IntegerField(blank=True, null=True)

bridge = models.IntegerField(blank=True, null=True)

pickups = models.IntegerField(blank=True, null=True)

info = models.TextField(blank=True)

def \_\_unicode\_\_(self):

return self.name

class Meta:

managed = True

db\_table = 'produser'

class Body(models.Model):

id = models.IntegerField(primary\_key=True)

material = models.CharField(max\_length=45, blank=True)

color = models.CharField(max\_length=45, blank=True)

type = models.CharField(max\_length=45, blank=True)

form = models.CharField(max\_length=45, blank=True)

def \_\_unicode\_\_(self):

return self.material

class Meta:

managed = True

db\_table = 'body'

class Bridge(models.Model):

id = models.IntegerField(primary\_key=True)

name = models.CharField(max\_length=45, blank=True)

material = models.CharField(max\_length=45, blank=True)

color = models.CharField(max\_length=45, blank=True)

produser = models.ForeignKey('Produser', related\_name='bridge\_produser', blank=True, null=True)

def \_\_unicode\_\_(self):

return self.name

class Meta:

managed = True

db\_table = 'bridge'

class Guitar(models.Model):

id = models.IntegerField(primary\_key=True)

name = models.CharField(max\_length=45, blank=True)

string\_amount = models.IntegerField(blank=True, null=True)

price = models.IntegerField(blank=True, null=True)

neck\_material = models.CharField(max\_length=45, blank=True)

fretboard\_material = models.CharField(db\_column='Fretboard\_material', max\_length=45, blank=True) # Field name made lowercase.

pick\_guard = models.NullBooleanField(db\_column='Pick\_guard', blank=True, null=True) # Field name made lowercase.

type = models.ForeignKey('GuitarType', db\_column='Type\_id', blank=True, null=True) # Field name made lowercase.

body = models.ForeignKey(Body, db\_column='Body\_id', blank=True, null=True) # Field name made lowercase.

bridge = models.ForeignKey(Bridge, db\_column='Bridge\_id', blank=True, null=True) # Field name made lowercase.

pickups = models.ForeignKey('Pickup', db\_column='Pickups\_id', blank=True, null=True) # Field name made lowercase.

guitar\_produser = models.ForeignKey('Produser', related\_name='guitar\_produser', db\_column='Guitar\_produser\_id', blank=True, null=True) # Field name made lowercase.

class Meta:

managed = True

db\_table = 'guitar'

class GuitarType(models.Model):

id = models.IntegerField(primary\_key=True)

name = models.CharField(max\_length=45, blank=True)

def \_\_unicode\_\_(self):

return self.name

class Meta:

managed = True

db\_table = 'guitar\_types'

class Pickup(models.Model):

id = models.IntegerField(primary\_key=True)

produser = models.ForeignKey('Produser', related\_name='pickups\_produser', blank=True, null=True)

type = models.CharField(max\_length=45, blank=True)

set\_type = models.CharField(max\_length=45, blank=True)

def \_\_unicode\_\_(self):

return self.set\_type

class Meta:

managed = True

db\_table = 'pickups'

class History(models.Model):

id = models.IntegerField(primary\_key=True)

trigger\_action = models.CharField(max\_length=45, blank=True)

guitar\_id = models.IntegerField(blank=True, null=True)

action\_time = models.DateTimeField(blank=True, null=True)

class Meta:

managed = False

db\_table = 'history'

class Variables(models.Model):

id = models.IntegerField(primary\_key=True)

name = models.CharField(max\_length=45, blank=True)

short\_name = models.CharField(max\_length=20, blank=True)

value = models.BooleanField(blank=True)

class Meta:

managed = False

db\_table = 'variables'

**Trigger:**

DELIMITER $$

USE `guitar\_schema`$$

DROP TRIGGER IF EXISTS `Guitar\_history\_insert`$$

CREATE TRIGGER `Guitar\_history\_insert` AFTER INSERT ON `Guitar` FOR EACH ROW

IF (SELECT value from `Variables` where name='TRIGGER\_GUITAR\_AFTER\_INSERT\_DISABLED')=0 THEN

INSERT INTO History(trigger\_action, guitar\_id, action\_time) VALUES('insert', NEW.id, NOW());

END IF;

$$

DROP TRIGGER IF EXISTS `Guitar\_history\_update`$$

CREATE TRIGGER `Guitar\_history\_update` AFTER UPDATE ON `Guitar` FOR EACH ROW

IF (SELECT value from `Variables` where name='TRIGGER\_GUITAR\_AFTER\_UPDATE\_DISABLED')=0 THEN

INSERT INTO History(trigger\_action, guitar\_id, action\_time) VALUES('update', OLD.id, NOW());

END IF;

$$

DROP TRIGGER IF EXISTS `Guitar\_history\_delete`$$

CREATE TRIGGER `Guitar\_history\_delete` BEFORE DELETE ON `Guitar` FOR EACH ROW

IF(SELECT value from `Variables` where name='TRIGGER\_GUITAR\_BEFORE\_DELETE\_DISABLED')=0 THEN

INSERT INTO History(trigger\_action, guitar\_id, action\_time) VALUES('delete', OLD.id, NOW());

END IF;

$$

DELIMITER ;

**Procedure:**

delimiter //

DROP PROCEDURE IF EXISTS statistics//

CREATE PROCEDURE statistics()

BEGIN

CREATE TEMPORARY TABLE IF NOT EXISTS res(

`id` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(45) NOT NULL,

`count` TINYINT(1) NOT NULL DEFAULT 0,

PRIMARY KEY (`id`));

INSERT INTO res(`name`, `count`) VALUES('Guitars', (SELECT COUNT(\*) FROM Guitar));

INSERT INTO res(`name`, `count`) VALUES('Pickups', (SELECT COUNT(\*) FROM Pickups));

INSERT INTO res(`name`, `count`) VALUES('Bridges', (SELECT COUNT(\*) FROM Bridge));

INSERT INTO res(`name`, `count`) VALUES('Bodies', (SELECT COUNT(\*) FROM Body));

INSERT INTO res(`name`, `count`) VALUES('Types', (SELECT COUNT(\*) FROM Guitar\_types));

INSERT INTO res(`name`, `count`) VALUES('Produsers', (SELECT COUNT(\*) FROM Produser));

INSERT INTO res(`name`, `count`) VALUES('History', (SELECT COUNT(\*) FROM History));

SELECT \* FROM res;

DROP TABLE res;

END//

delimiter ;

**Event:**

delimiter //

DROP EVENT IF EXISTS account//

CREATE EVENT account

ON SCHEDULE EVERY 1 WEEK

DO

BEGIN

INSERT INTO `guitar\_schema`.`Accounting`(`date`, `Guitars`, `Pickups`, `Bridges`, `Bodies`, `Types`, `Produsers`, `History`)

VALUES(NOW(),

(SELECT COUNT(\*) FROM Guitar),

(SELECT COUNT(\*) FROM Pickups),

(SELECT COUNT(\*) FROM Bridge),

(SELECT COUNT(\*) FROM Body),

(SELECT COUNT(\*) FROM Guitar\_types),

(SELECT COUNT(\*) FROM Produser),

(SELECT COUNT(\*) FROM History));

END //

delimiter ;

**Views:**

def index(request):

table = tables.GuitarTable(models.Guitar.objects.all())

RequestConfig(request).configure(table)

return render(request, 'guitar\_app/index.html', {'table': table})

def action(request):

if request.method == 'POST':

action = request.POST.get('action', False)

if action:

pks = request.POST.getlist("selection")

#selected\_objects = models.Guitar.objects.filter(pk\_\_in=pks)

if action == 'delete':

models.Guitar.objects.filter(pk\_\_in=pks).delete()

if action == 'history':

return HttpResponseRedirect('/history/')

if action == 'statistics':

return HttpResponseRedirect('/statistics/')

if action == 'load':

return HttpResponseRedirect('/load\_data\_from\_file/')

return HttpResponseRedirect('/')

return HttpResponse('no POST in action view')

def add(request):

if request.method == 'POST':

form = forms.GuitarAddForm(request.POST)

if form.is\_valid():

guitar = form.save()

return HttpResponseRedirect('/')

else:

return HttpResponse('Form is not valid')

else:

form = forms.GuitarAddForm()

return render(request,'guitar\_app/add.html', {'form': form})

def guitar\_detail(request, pk):

if request.method == 'POST':

form = forms.GuitarAddForm(request.POST)

if form.is\_valid():

guitar = models.Guitar.objects.get(pk=pk)

t = forms.GuitarAddForm(request.POST, instance=guitar)

t.save()

return HttpResponseRedirect('/')

else:

form = forms.GuitarAddForm(instance=models.Guitar.objects.get(pk=pk))

return render(request, 'guitar\_app/guitar\_detail.html', {'form': form, 'pk': pk})

def load\_data\_view(request):

sql\_scripts.load\_data\_from\_files()

return HttpResponseRedirect('/')

def history\_action(request):

if request.method == 'POST':

VarFormSet = modelformset\_factory(models.Variables, form=forms.VariablesForm, extra=0)

formset = VarFormSet(request.POST)

for form in formset:

form.save()

return HttpResponseRedirect('/')

def history(request):

VarFormSet = modelformset\_factory(models.Variables, form=forms.VariablesForm, extra=0)

formset = VarFormSet()

table = tables.HistoryTable(models.History.objects.all())

RequestConfig(request).configure(table)

return render(request, 'guitar\_app/history.html', {'table': table, 'formset': formset})

def statistics(request):

table = tables.StatisticsTable(sql\_scripts.get\_statistics())

RequestConfig(request).configure(table)

return render(request, "guitar\_app/statistics.html", {"table": table})

