Реляционные базы данных

Решаемые проблемы

- Структура хранения
- Эффективный поиск данных
- Управление памятью
- Совместный доступ к данным
- Атомарные операции - транзакции
- Язык управления базой и данными - **SQL**

Реляционная модель данных

Данные хранятся в виде таблиц. У каждой таблицы фиксированное число столбцов. Все данные в столбце одного типа.

blog_post

id	title	category_id
1	Python	1
3	Python for dummies	1
15	Perl	2
7	C++ in 21 day	3

blog_category

id	title	auditory
1	Python	4000
2	Perl	2000
3	C++	10000
4	Java	8000

Проектирование базы данных

Основная задача проектирования - сокращение избыточности и дублирования данных.

Существуют формальные правила проверки схемы базы данных на «правильность» - **нормальные формы** базы данных.

Проектирование на практике

- Логическое разделение сущностей
- Выделение синтетических первичных ключей
- Связи 1:N, N:1 реализуются через внешний ключ
- Связи 1:1 реализуются через внешний ключ с unique index
- Связи N:M реализуются через промежуточную таблицу
- Атрибут с фиксированным числом значений – внешняя таблица либо поле типа enum

Базы данных в Python

Подключение к базе

Полное интерфейса для работы с СУБД в PEP-0249

```
import MySQLdb
connection = MySQLdb.connect(host="localhost", user="joe",
                     passwd="moonpie", db="thangs")
cursor = connection.cursor()
connection.begin()
# запросы с использованием cursor
connection.commit()
connection.close()
```

Выполнение запросов

```
cursor.execute("""
    update users set age = age + 1 where name = %s
""", (name,))

cursor.execute("select * from users")
users = cursor.fetchall()

cursor.execute("""
     select * from users where name = %s
""", (name,))
user = cursor.fetchone()
```

Вставка многих записей

```
cursor.executemany(
        "INSERT INTO users (name, age) VALUES (%s, %s)",
        [
            ("Igor", 18 ),
            ("Petr", 16 ),
            ("Dasha", 17 )
        ]
)
```

Placeholders

```
email = "' OR '1'='1"
cursor.execute(
    "SELECT * FROM users WHERE email = '" + email + "'"
)
# SELECT * FROM users WHERE email = '' OR '1'='1'

cursor.execute(
    "SELECT * FROM users WHERE email = '%s'",
    email
)
```

Базы данных в Django

Прямой доступ к базе

```
from django.db import connection, connections

cur = connection.cursor()
cur.execute("select * from tbl limit 10")

default_cur = connections["default"].cursor()
default_cur.execute("select * from tbl2 limit 10")
another_cur = connections["another"].cursor()
another_cur.execute("select * from tbl2 limit 10")

# settings.py
DATABASES = {
    "default": {"ENGINE": ... },
    "another": { ... }
}
```

Полезные утилиты

- `./manage.py check` - проверить приложения
- `./manage.py makemigrations` - создать миграции
- `./manage.py migrate` - накатить миграции
- `./manage.py shell` - запустить python shell
- `./manage.py dbshell` - запустить клиент базы данных

Django Models

ORM - Object relational mapping - библиотек предоставляющая объектно-ориентированный интерфейс к реляционной базе данных. **Django Models** - библиотека ORM в Djnago.

Django	**SQL**
Класс Model	Таблица
Объект Model	Строка таблицы
QuerySet	Запрос

ORM vs SQL

```
cursor.execute("select * from users where age > 18")
for user in cursor.fetchall():
    pk, name, age = user
    print(name)

for user in User.objects.filter(age__gt=18):
    print(user.name)
```

Модели Django

```
from django.db import models
class Post(models.Model):
    title = models.CharField(max_length=255)
    content = models.TextField()
    creation_date = models.DateTimeField(blank=True)
    def __str__(self):
        return self.title
    def get_absolute_url(self):
        return "/post/{}/".format(self.pk)
    class Meta:
        db_table = "blogposts"
        ordering = ["-creation_date"]
```

Типы полей

Django	**MySQL**
CharField	VARCHAR(N)
EmailField	
TextField	LONGTEXT
BooleanField	TINYINT(1)
IntegerField	INT(11)
DateField	DATE
DateTimeField	DATETIME

Свойства полей

- `blank` - поле может быть пустым
- `null` - поле допускает в базе значение NULL
- `max_length` - максимальная длина поля
- `primary_key` - поле - первичный ключ
- `unique` - для поля нужен уникальный индекс в базе
- `db_index` - для поля нужен индекс в базе
- `default` - значение по-умолчанию
- `choices` - варианты значений

Использование Choices

```
from django.db import models

class Post(models.Model):
    POST_CHOICES = [
        ("NE", "News"),
        ("AR", "Article"),
    ]
    title = models.CharField(max_length=255)
    post_type = models.CharField(max_length=2, choices=POST_CHOICES)
```

Связи между моделями

Связи между моделями

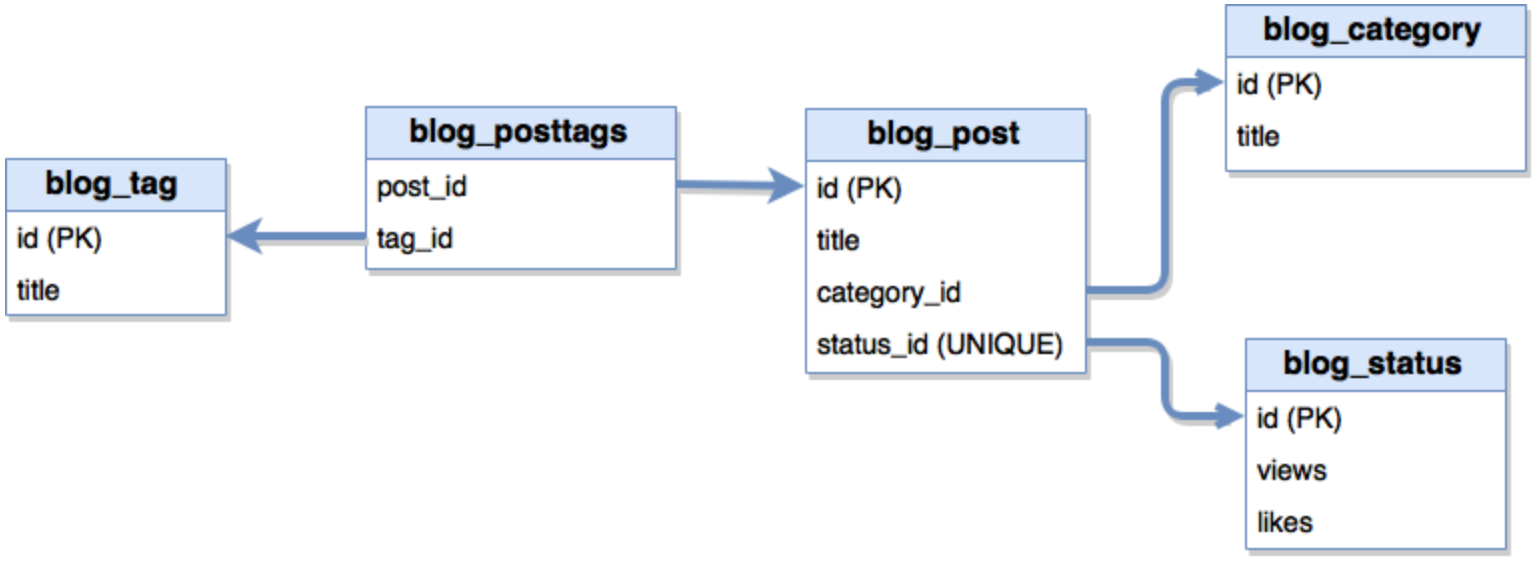
```
class Post(models.Model):
    title = models.CharField(max_length=255)
    # еще поля...

    category = models.ForeignKey(Category,
        null=True, on_delete=models.SET_NULL)

    status = models.OneToOneField(PostStatus,
        on_delete=models.CASCADE)

    tags = models.ManyToManyField(Tag)
```

Реализация в СУБД



Ограничения внешних ключей

Применимо к полям типа `ForeignKey`, `OneToOneField`

- `RESTRICT` → `models.PROTECT`
- `CASCADE` → `models.CASCADE`
- `SET NULL` → `models.SET_NULL`
- `NO ACTION` → `models.DO_NOTHING`

Использование отношений в коде

```
# прямое использование
post = Post.objects.get(pk=1)
category = post.category          # Category
category_id = post.category_id    # int
status = post.status              # Status
status_id = post.status_id        # int
tags_manager = post.tags          # RelatedManager
post.tags.all()                   # [ Tags ]
# использование обратного отношения
# ForeignKey.related_name
# ManyToManyField.related_name
category.post_set.all()           # [ Post ]
tag.post_set.all()                # [ Post ]
```