Готовим **SQLAlchemy** с умом

Положительные черты

- Скорость
- Кастомность
- Три варианта синтаксиса
- Множество расширений
- Независимость от фреймверка
- Возможность использования специальных полей
- Работа с большим количеством движков
- Наличие асинхронного табличного синтаксиса
- Отличная поддержка и шикарная документация

Отрицательные черты

- Высокий порог входа
- Нет и не будет асинхронного деклоратива
- Сложность кода для некоторых операций
- Высокая сложность исходного кода
- Необходимо использовать двойной синтаксис
- Требует большего понимания устройства БД

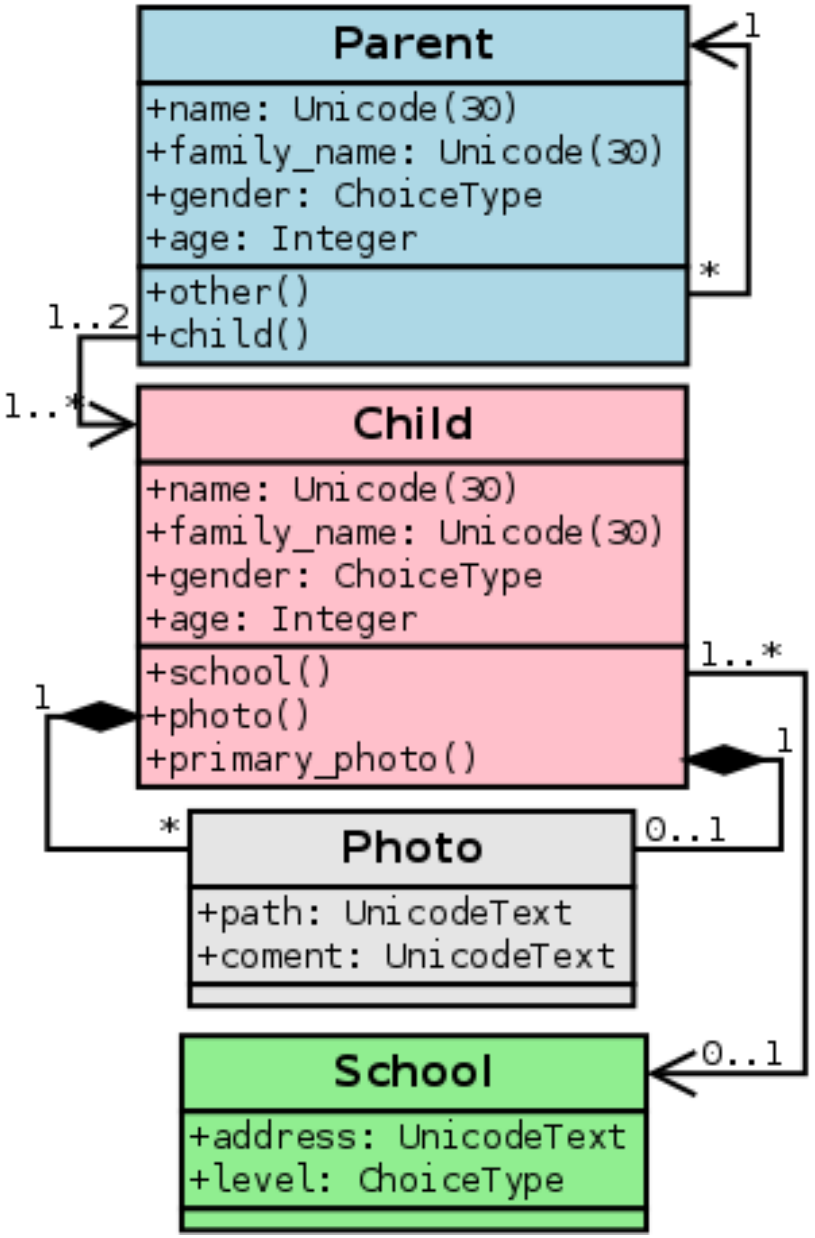
Связи: OTM

```
class Child(Base):

    photo = relationship("Photo",
            foreign_keys="Photo.child_id",
            backref=backref(
                "photo_child",
                order_by=family_name
            ),
            cascade="all,
                    delete, delete-orphan"
     )

class Photo(Base):

    child_id = Column(Integer,
            ForeignKey("child.id")
     )
```

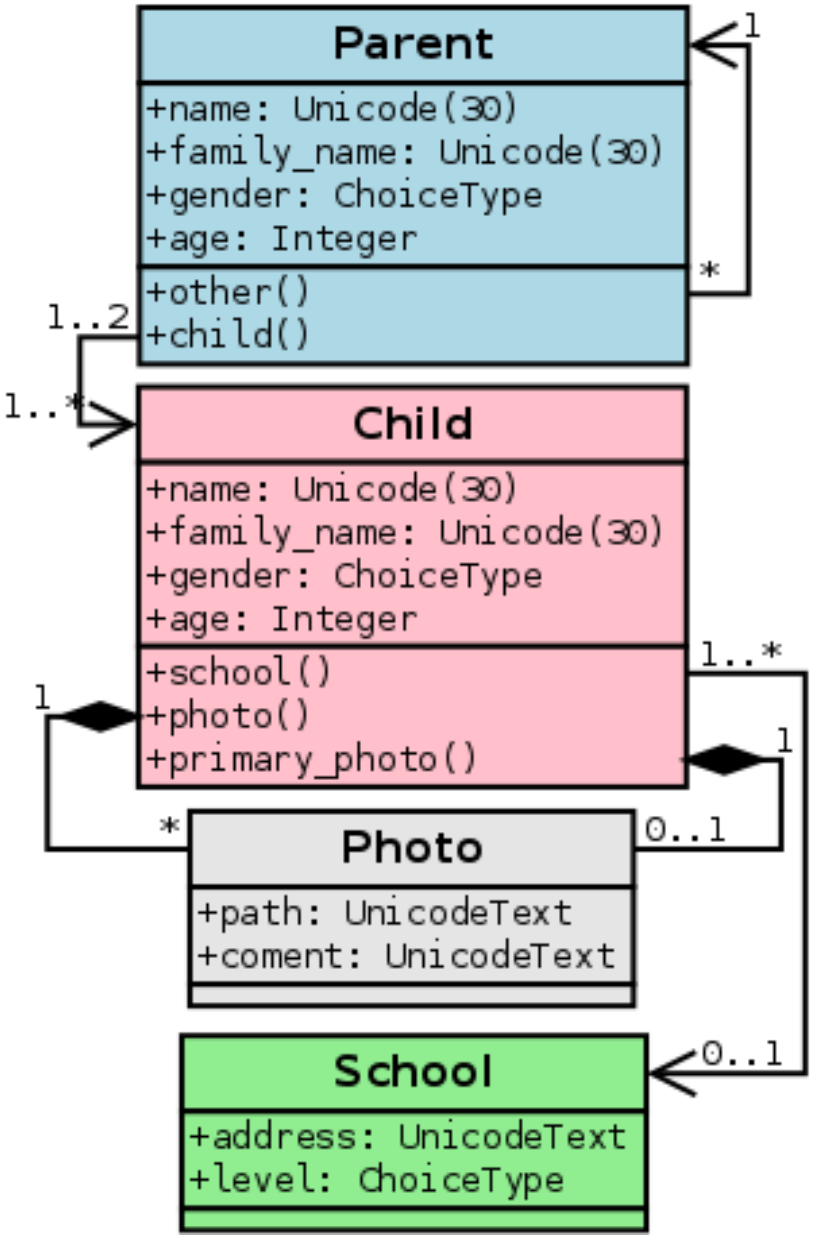


Связи: MTO

```
class Child(Base):

    school_id = Column(Integer,
        ForeignKey("school.id")
    )

    school = relationship("School",
        backref=backref("pupil",
        order_by=family_name)
    )
```



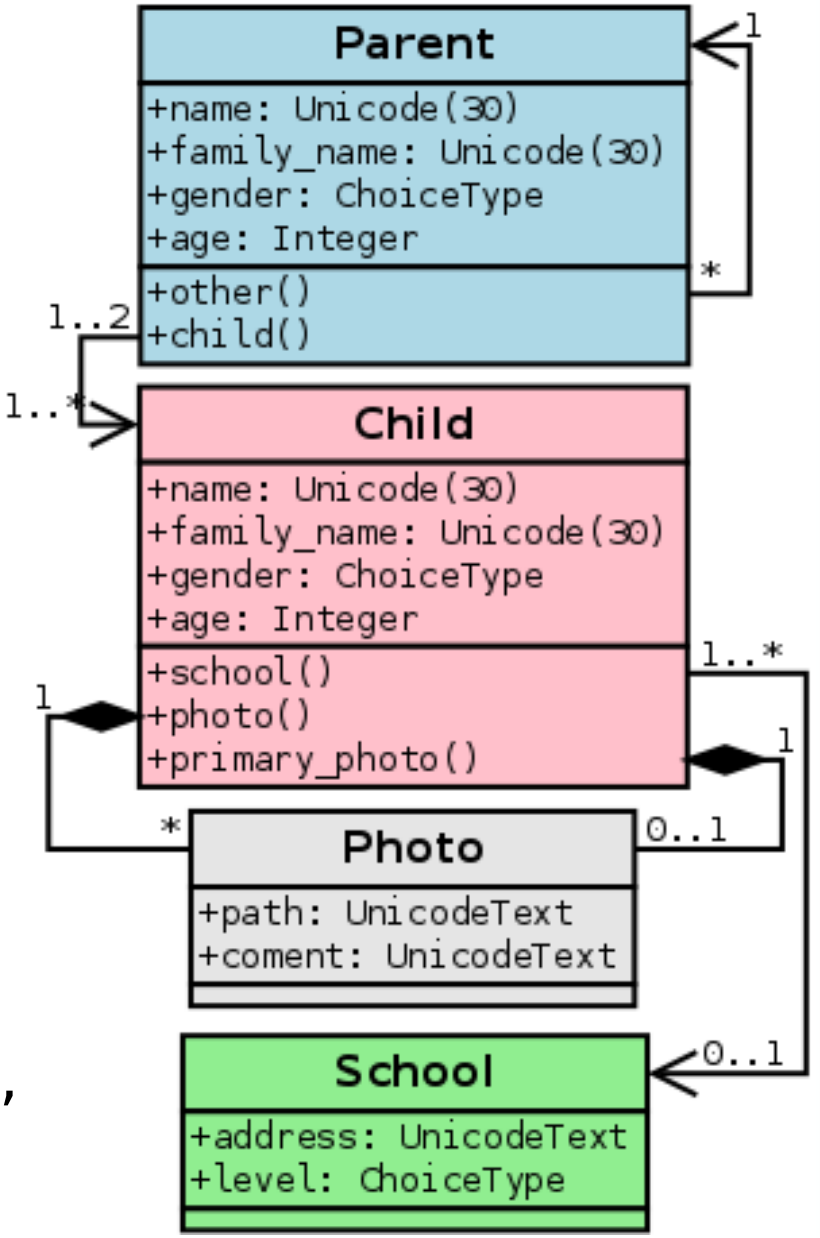
Связи: OTO

```
class Child(Base):

    primary_photo = relationship(
        "Photo",
        foreign_keys="Photo.
            primary_child_id",
        uselist=False,
    )

class Photo(Base):

    primary_child_id = Column(Integer,
        ForeignKey("child.id")
    )
```



Связи: MTM

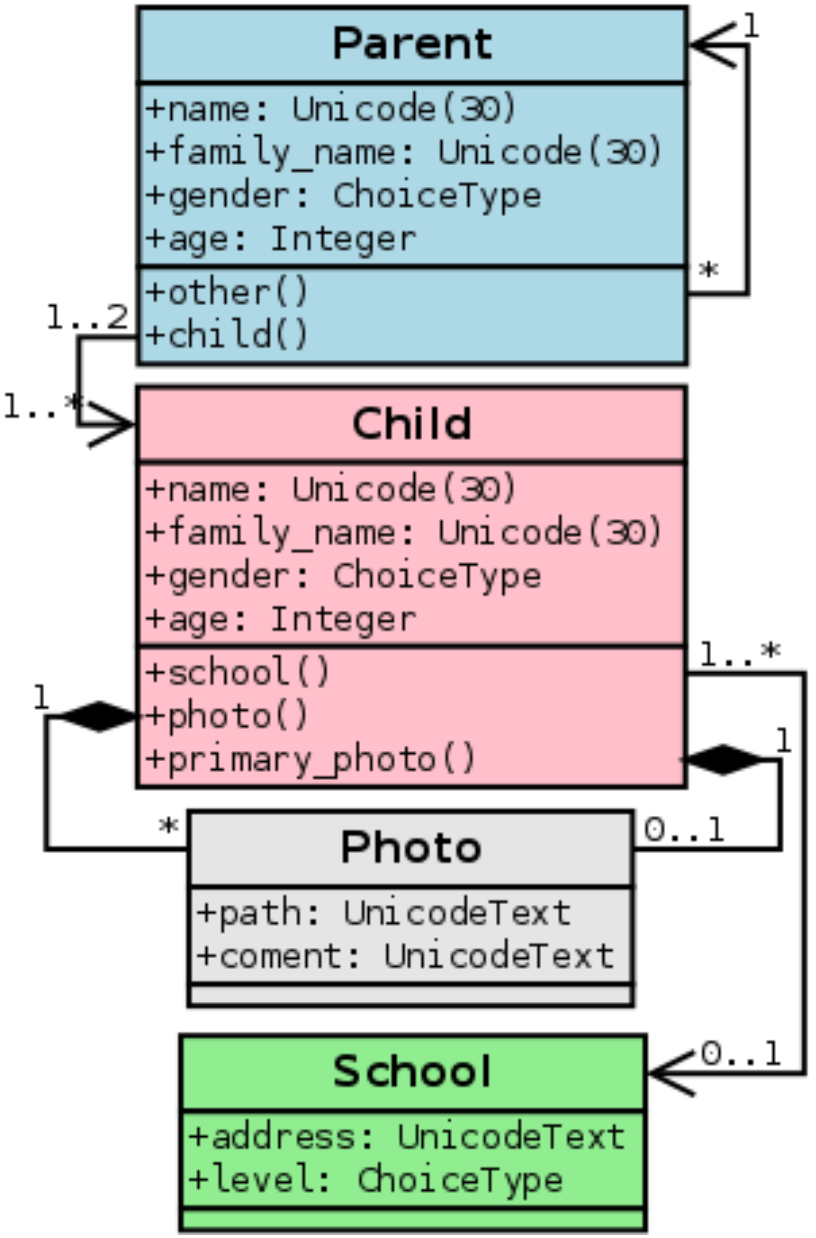
```
parent_child = Table("parent__child",
    Base.metadata,

    Column("parent_id", Integer,
                ForeignKey("parent.id"),
                primary_key=True),

    Column("child_id", Integer,
                ForeignKey("child.id"),
                primary_key=True)
)

class Parent(Base):

    child = relationship("Child",
        secondary=parent_child,
    )
```

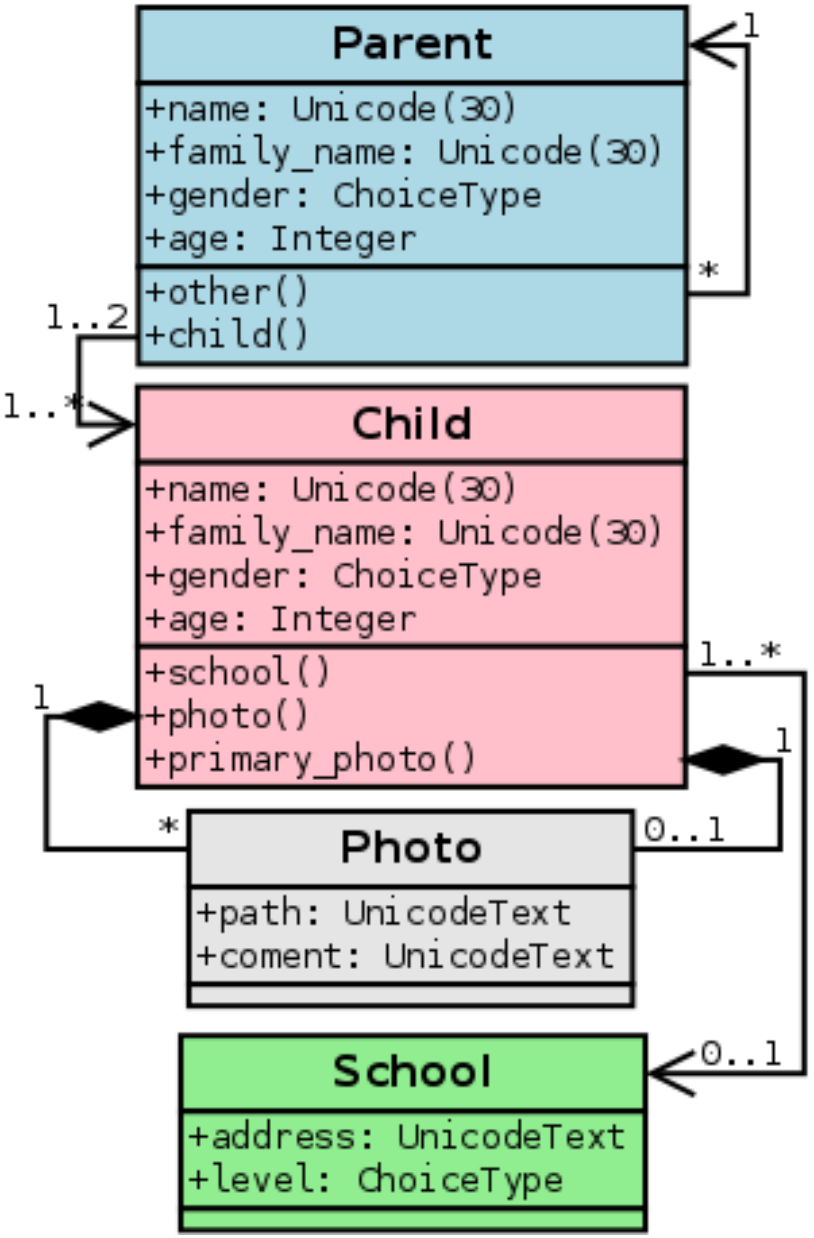


Связи: Self

```
class Parent(Base):

    other_id = Column(Integer,
        ForeignKey("parent.id")
    )

    other = relationship("Parent",
        remote_side=[id],
        backref=backref("main
            order_by=family_name)
        )
    )
```



Создание фабрики сессий

```
engine_str = "postgresql+psycopg2://{}:{}@{}/{}"\
    .format(user, password, db_host, db_name)

engine = create_engine(engine_str,
                       convert_unicode=True,
                       echo=False)

Base.metadata.create_all(engine)

maker = sessionmaker(autocommit=False,
                     autoflush=False,
                     bind=engine)

fabric_session = scoped_session(maker)
```

Очень простые запросы

```
child = db_session.query(Child).first()

child = db_session.query(Child).get(3)

children = db_session.query(Child).all()

children = db_session.query(Child).limit(10).offset(5).all()

children = db_session.query(Child).order_by(Child.age).all()

children = db_session.query(Child.gender,
                            func.count(Child.gender))
                            .group_by(Child.gender).all()
```

Запросы с фильтрацией

```
children = db_session.query(Child)\
    .filter_by(gender="m").all()

children = db_session.query(Child)\
    .filter(Child.gender != "w").all()

children = db_session.query(Child)\
    .filter(Child.name.like("%5%")).all()

children = db_session.query(Child) \
    .filter(Child.age > 15)\
    .filter(Child.gender == "w").all()

children = db_session.query(Child)\
   .filter(and_(Child.age > 15,
                Child.gender == "w")).all()
```

Запросы с фильтрацией по внешним связям

```
children = db_session.query(Child)\
    .join(School)\
    .filter(School.level == "g").all()

children = db_session.query(Child)\
    .join(Child.photo)\
    .filter(Photo.path.like("%2")).all()

children = db_session.query(Child)\
    .outerjoin(Child.photo)\
    .filter(Photo.path.like("%2")).all()

children = db_session.query(Child)\
    .join(Photo, Child.photo).all()
```

Eager Loading query

```
children = db_session.query(Child)\
    .options(joinedload(Child.school)).all()

children = db_session.query(Child)\
    .options(subqueryload(Child.photo)).all()

parents = db_session.query(Parent)\
    .options(subqueryload(Parent.child))\
    .options(joinedload(Parent.child, Child.school)).all()

children = db_session.query(Child)\
    .join(Child.photo) \
    .filter(Photo.path.like("%3")) \
    .options(contains_eager(Child.photo)).all()
```

Фокусы: query_property()

```
Session = scoped_session(sessionmaker())
BaseProperty = type("BaseMixin",
                    (object,),
                    {"query": Session.query_property()})
Base = declarative_base(cls=BaseProperty)


Child.query.filter(Child.id > 10).all()
Parent.query.filter(Parent.age < 40).all()


# create session
Base.metadata.bind = engine
Base.metadata.create_all()
Session.configure(bind=engine)
```

Фокусы: hybrid_property

```
class Parent(Base):

    age = Column(Integer)

    @hybrid_property
    def is_old_man(self):
        return and_(self.age > 40, self.gender == "m")


# query
parents = db_session.query(Parent)\
    .filter(Parent.is_old_man).all()
```

Фокусы: hybrid expression

```
class School(Base):

    @hybrid_property
    def count_pupil(self):
        return len(self.pupil)

    @count_pupil.expression
    def count_pupil(cls):
        return select([func.count(Child.id)]) \
                .where(Child.school_id == cls.id) \
                .label("count_pupil")

# query
schools = db_session.query(School)\
    .filter(School.count_pupil > 40).all()
```

Фокусы:

yield_per, load_only, union

```
for children in db_session.query(Child).yield_per(100): pass

parents = db_session.query(Parent)\
    .options(
        load_only("family_name"),
        joinedload(Parent.child).load_only("family_name")
    ).all()

children_m = db_session.query(Child)\
    .filter(Child.gender == "m")
children_w = db_session.query(Child)\
    .filter(Child.gender == "w")
result = children_m.union(children_w)\
    .filter(Child.age > 12).all()
```

Фокусы: event and inspect

```
from sqlalchemy import event, inspect

@event.listens_for(Photo, "before_update")
def check_child_age(mapper, connection, target):

    parameters = set(inspect(target).dict.keys())
    parameters.remove("_sa_instance_state")
    un_change = inspect(target).unmodified
    change = parameters.difference(un_change)
    has_comment = hasattr(target, "comment")

    if change.difference({"path"}) and has_comment:
        setattr(target, "comment", "path is changed...")
```

Фокусы: collection_class, JSONB

```
class Parent(Base):

    child = relationship("Child",
                         secondary=parent_child,
                         backref=backref("parent"),
                         collection_class=set)

    extra = Column(JSONB)


Parent.extra = {"work": "employe"}
parents = db_session.query(Parent)\
    .filter(Parent.extra["work"].cast(Unicode) == "employee")
```

Фокусы: subquery(), cte()

```
photo = db_session.query(Photo)\
    .filter(Photo.path.like("%2"))
smt = photo.subquery()
children = db_session.query(Child)\
    .join(smt, Child.id == smt.c.child_id).all()
```

```
child = db_session.query(Child)\
    .filter(Child.age > 10)
smt = child.cte()
children = db_session.query(smt)\
    .filter(Child.gender == "m").all()
```

Фокусы: Association Proxy

```
class User(Base):
    name = Column(String(64))

    keywords = association_proxy('user_keywords', 'keyword')

class UserKeyword(Base):
    user_id = Column(Integer, ForeignKey('user.id'), primary_key=True)
    keyword_id = Column(Integer, ForeignKey('keyword.id'),
                                    primary_key=True)
    special_key = Column(String(50))

    keyword = relationship("Keyword")
    user = relationship(User, backref=backref("user_keywords",
                                cascade="all, delete-orphan"))

class Keyword(Base):
    keyword = Column('keyword', String(64))
```

Фокусы: Association Proxy

```
user = User('log')
user.keywords.append(Keyword('first keyword'))
user.keywords.append(Keyword('second keyword'))

user.user_keywords\
    .append(UserKeyword(Keyword('third keyword')))

UserKeyword(Keyword('last_keyword'),
                    user,
                    special_key='special key')
```

Неожиданно

«тяжелые» запросы

```
parents = parents = db_session.query(Parent)\
    .filter(School.level == "g").all()

class Child(Base):

    @hybrid_property
    def child_medium_age(self):
        if self.child:
            return sum([c.age for c in self.child]) /
                                                len(self.child)

parents = db_session.query(Parent).limit(30).all()
for parent in parents:
    print(parent.child_medium_old)
```

Нюансы: Автодобавление связанных объектов

```
child = db_session.query(Child)\
    .options(subqueryload(Child.photo)).get(1)
child.photo.append(Photo(path="/image/image.jpg"))

db_session.commit()

scoped_session.remove()

child = db_session.query(Child)\
    .options(subqueryload(Child.photo)).get(1)
for photo in child.photo:
    print(photo.path)
```

Наследование: __abstract__

```
class Human(Base):
    __abstract__ = True
    id = Column(Integer, primary_key=True)
    name = Column(Unicode(30), nullable=False)
    family_name = Column(Unicode(30), nullable=False)
    age = Column(Integer)

class Parent(Human):
     school_id = Column(Integer, ForeignKey("school.id"))
     school = relationship("School", backref=backref("pupil"))

class Child(Human):
     child = relationship("Child",
          secondary=parent_child,
          backref=backref("parent", order_by=family_name)
     )
```

Наследование: polymorphic

```
class Employee(Base):
    __tablename__ = 'employee'
    id = Column(Integer, primary_key=True)
    type = Column(String(20))
    __mapper_args__ = {'polymorphic_on':type,
                       'polymorphic_identity':'employee',
                       'with_polymorphic':'*'}

class Engineer(Employee):
    __tablename__ = 'engineer'
    id = Column(Integer, ForeignKey('employee.id'), primary_key=True)
    __mapper_args__ = {'polymorphic_identity':'engineer'}

class Manager(Employee):
    __tablename__ = 'manager'
    id = Column(Integer, ForeignKey('employee.id'), primary_key=True)
    __mapper_args__ = {'polymorphic_identity':'manager'}
```

Советы алхимика

- Не используйте доступ к внешним данным в hybrid-***.
- Не делайте фильтра по внешним данным без явного их подключения (join или outerjoin).
- Не забывайте про «нетерпеливые» (eager) загрузки.
- Всегда используйте экземпляр scoped_session и не забывайте делать scoped_session.remove()
- Разделяйте структуры данных и методы их выполнения (__abstract__ или доступ к атрибуту класса).
- Не увлекайтесь Mixin-технологиями. Иногда чтобы поддержать Mixin нужно вместо обычной OTM или MTO связи «ваять» MTM.
- Не «кастомизируйте» модели, если есть варианты альтернативного использования и тем более, когда это можно сделать запросом.
- Используйте полиморфическое наследование только когда оно действительно нужно и без него никак.
- **Читайте SQL!!!**

Расширения SQLAlchemy

- Alembic
- SQLAlchemy-Utils
- Wtforms-Alchemy

Спасибо за внимание.

Без вопросов не уйду!!!