

Tinkoff python

Лекция 5

Реляционные базы данных



Перминов Сергей

Backend Dialog System

dialog platform for bots



Базы данных - зачем?

Базы данных - зачем?

Чтобы хранить данные :)

Базы данных - зачем?

Чтобы хранить данные :)

Писать свою реализацию хранилища
сложно и долго.

Базы данных - зачем?

Чтобы хранить данные :)

Писать свою реализацию хранилища
сложно и долго.

Реляционные и нереляционные.

Реляционные базы данных

Используем, когда данные хорошо структурированы, и нам важны связи между ними.

Данные хранятся в таблицах

task

| title | created_at | deadline | done_at |
|------------------------------|------------|----------|---------|
| расширить API | 25.02 | 03.03 | 01.03 |
| обновиться до python 3.8 | 03.03 | 15.03 | NULL |
| добавить логов в http-клиент | 21.02 | NULL | NULL |

task

| title | ... | author | assigned_to | project |
|--------------------------|-----|--------|-------------|---------|
| расширить API | ... | Даниил | Антон | chatbot |
| обновиться до python 3.8 | ... | Даниил | NULL | talkbot |

task

| title | ... | author | assigned_to | project |
|--------------------------|-----|--------|-------------|---------|
| расширить API | ... | Даниил | АНТОН | chatbot |
| обновиться до python 3.8 | ... | Даниил | NULL | talkbot |

user

| name | role | ... |
|--------|-----------------|-----|
| Даниил | project manager | ... |
| АНТОН | backend dev | ... |

task

| title | ... | author | assigned_to | project |
|--------------------------|-----|--------|-------------|---------|
| расширить API | ... | Даниил | АНТОН | chatbot |
| обновиться до python 3.8 | ... | Даниил | NULL | talkbot |

user

| name | role | ... |
|--------|-----------------|-----|
| Даниил | project manager | ... |
| АНТОН | backend dev | ... |

project

| title | state | ... |
|---------|---------------|-----|
| chatbot | a b testing | ... |
| talkbot | alpha version | ... |

| title | ... | project | project_version |
|------------------------------|------------|----------------|------------------------|
| расширить API | ... | chatbot | a b testing |
| обновиться до python 3.8 | ... | talkbot | alpha version |
| добавить логов в http-клиент | ... | talkbot | alpha version |

| title | ... | project | project_version |
|------------------------------|-----|----------------|------------------------|
| расширить API | ... | chatbot | a b testing |
| обновиться до python 3.8 | ... | talkbot | alpha version |
| добавить логов в http-клиент | ... | talkbot | alpha version |

| title | ... | project | project_version |
|------------------------------|-----|----------------|------------------------|
| расширить API | ... | chatbot | a b testing |
| обновиться до python 3.8 | ... | talkbot | production |
| добавить логов в http-клиент | ... | talkbot | alpha version |

| title | ... | project | project_version |
|------------------------------|------------|----------------|------------------------|
| расширить API | ... | chatbot | a b testing |
| обновиться до python 3.8 | ... | talkbot | alpha version |
| добавить логов в http-клиент | ... | talkbot | alpha version |

| title | ... | project | project_version |
|------------------------------|------------|----------------|------------------------|
| расширить API | ... | chatbot | a b testing |
| обновиться до python 3.8 | ... | talkbot | production |
| добавить логов в http-клиент | ... | talkbot | alpha version |

One to one

страна - столица

человек - паспортные данные

One to many

One to many

task

| title | author | ... |
|------------------------------|--------|-----|
| расширить API | Даниил | ... |
| обновиться до python 3.8 | Даниил | ... |
| добавить логов в http-клиент | Антон | |

user

| name | role | ... |
|--------|-----------------|-----|
| Даниил | project manager | ... |
| Антон | backend dev | ... |

Many to many

Many to many

film

| title | year | ... |
|--------------|------|-----|
| Джентельмены | 2019 | ... |
| 1917 | 2019 | ... |
| Паразиты | 2019 | ... |

user

| login | registered_at | ... |
|-------|---------------|-----|
| foo | 05.05.2013 | ... |
| bar | 26.12.2019 | ... |

Many to many

film

| title | year | ... |
|--------------|-------------|------------|
| Джентельмены | 2019 | ... |
| 1917 | 2019 | ... |
| Паразиты | 2019 | ... |

user

| login | registered_at | ... |
|--------------|----------------------|------------|
| foo | 05.05.2013 | ... |
| bar | 26.12.2019 | ... |

film_rating

| film | user | rating |
|--------------|-------------|---------------|
| Джентельмены | foo | 9 |
| Джентельмены | bar | 6 |
| 1917 | bar | 8 |

Первичный ключ

Первичный ключ

- позволяет однозначно идентифицировать строку;

Первичный ключ

- позволяет однозначно идентифицировать строку;
- должен быть отвязан от реальных сущностей.

Первичный ключ

- позволяет однозначно идентифицировать строку;
- должен быть отвязан от реальных сущностей.

| email | password_hash | registered_at | ... |
|--------------|----------------------|----------------------|------------|
| foo@kek.ru | 2cf24dba5f... | 05.05.2013 | ... |
| bar@kek.ru | 58756879c0... | 26.12.2019 | ... |

Первичный ключ

| email | password_hash | registered_at | ... |
|--------------|----------------------|----------------------|------------|
| foo@kek.ru | 2cf24dba5f... | 05.05.2013 | ... |
| bar@kek.ru | 58756879c0... | 26.12.2019 | ... |

Первичный ключ

| email | password_hash | registered_at | ... |
|--------------|----------------------|----------------------|------------|
| foo@kek.ru | 2cf24dba5f... | 05.05.2013 | ... |
| bar@kek.ru | 58756879c0... | 26.12.2019 | ... |

| email | registered_at | ... | facebook_id |
|--------------|----------------------|------------|--------------------|
| foo@kek.ru | 05.05.2013 | ... | NULL |
| bar@kek.ru | 26.12.2019 | ... | NULL |
| NULL | 10.03.2020 | ... | 14446842256 |

Первичный ключ

| email | password_hash | registered_at | ... |
|--------------|----------------------|----------------------|-----|
| foo@kek.ru | 2cf24dba5f... | 05.05.2013 | ... |
| bar@kek.ru | 58756879c0... | 26.12.2019 | ... |

| email | registered_at | ... | facebook_id |
|--------------|----------------------|-----|--------------------|
| foo@kek.ru | 05.05.2013 | ... | NULL |
| bar@kek.ru | 26.12.2019 | ... | NULL |
| NULL | 10.03.2020 | ... | 14446842256 |

| id | email | ... | facebook_id |
|-----------|--------------|-----|--------------------|
| 1 | foo@kek.ru | ... | NULL |
| 2 | bar@kek.ru | ... | NULL |
| 3 | NULL | ... | 14446842256 |

Да кто такой этот sql?

Structured Query Language

<https://www.w3schools.com/sql/>

<http://www.sql-tutorial.ru/ru/content.html>

create table

```
create table film (  
    id serial primary key,  
    title varchar not null,  
    budget integer default null,  
    premiere date not null  
);
```

Колонки типизированы.

* все примеры на PostgreSQL 11.6

create table

```
create table film (  
    id serial primary key,  
    title varchar not null,  
    budget integer default null,  
    premiere date not null  
);
```

Колонки типизированы.

- целые числа, строки, даты...

* все примеры на PostgreSQL 11.6

create table

```
create table film (  
    id serial primary key,  
    title varchar not null,  
    budget integer default null,  
    premiere date not null  
);
```

Колонки типизированы.

- целые числа, строки, даты...
- json, array, enum;

* все примеры на PostgreSQL 11.6

create table

```
create table film (  
    id serial primary key,  
    title varchar not null,  
    budget integer default null,  
    premiere date not null  
);
```

Колонки типизированы.

- целые числа, строки, даты...
- json, array, enum;
- ...
- геометрические примитивы (некоторые БД).

* все примеры на PostgreSQL 11.6

insert

```
insert into film (title, budget, premiere)
  values ( 'Джентельмены', 22000000, '2019-12-03' );
```

insert

```
insert into film (title, budget, premiere)
  values ('Джентельмены', 22000000, '2019-12-03');
```

```
insert into film (title, budget, premiere)
  values
    ('1917', 100000000, '2019-12-04'),
    ('Паразиты', NULL, '2019-05-21');
```

select

```
> select * from film;
```

| id | title | budget | premiere |
|----|--------------|-----------|------------|
| 1 | Джентельмены | 22000000 | 2019-12-03 |
| 2 | 1917 | 100000000 | 2019-12-04 |
| 3 | Паразиты | | 2019-05-21 |

select

```
> select * from film;
```

| id | title | budget | premiere |
|----|--------------|-----------|------------|
| 1 | Джентельмены | 22000000 | 2019-12-03 |
| 2 | 1917 | 100000000 | 2019-12-04 |
| 3 | Паразиты | | 2019-05-21 |

```
> select title, premiere from film;
```

| title | premiere |
|--------------|------------|
| Джентельмены | 2019-12-03 |
| 1917 | 2019-12-04 |
| Паразиты | 2019-05-21 |

select

```
> select * from film where budget > 50000000;
```

| id | title | budget | premiere |
|----|-------|-----------|------------|
| 2 | 1917 | 100000000 | 2019-12-04 |

select

```
> select * from film where budget > 50000000;
```

| id | title | budget | premiere |
|----|-------|-----------|------------|
| 2 | 1917 | 100000000 | 2019-12-04 |

```
> select * from film order by premiere desc;
```

| id | title | budget | premiere |
|----|--------------|-----------|------------|
| 2 | 1917 | 100000000 | 2019-12-04 |
| 1 | Джентельмены | 22000000 | 2019-12-03 |
| 3 | Паразиты | | 2019-05-21 |

update

```
> update film
    set budget = budget + 100000
  where id = 2;
```

UPDATE 1

```
> select * from film;
```

| id | title | budget | premiere |
|----|--------------|-----------|------------|
| 1 | Джентельмены | 22000000 | 2019-12-03 |
| 3 | Паразиты | | 2019-05-21 |
| 2 | 1917 | 101000000 | 2019-12-04 |

delete

```
> select * from film;
```

| id | title | budget | premiere |
|----|--------------|-----------|------------|
| 1 | Джентельмены | 22000000 | 2019-12-03 |
| 3 | Паразиты | | 2019-05-21 |
| 2 | 1917 | 101000000 | 2019-12-04 |

delete

```
> select * from film;
```

| id | title | budget | premiere |
|----|--------------|-----------|------------|
| 1 | Джентельмены | 22000000 | 2019-12-03 |
| 3 | Паразиты | | 2019-05-21 |
| 2 | 1917 | 101000000 | 2019-12-04 |

```
> delete from film where id = 2;
```

```
DELETE 1
```

delete

```
> select * from film;
```

| id | title | budget | premiere |
|----|--------------|-----------|------------|
| 1 | Джентельмены | 22000000 | 2019-12-03 |
| 3 | Паразиты | | 2019-05-21 |
| 2 | 1917 | 101000000 | 2019-12-04 |

```
> delete from film where id = 2;
```

```
DELETE 1
```

```
> select * from film;
```

| id | title | budget | premiere |
|----|--------------|----------|------------|
| 1 | Джентельмены | 22000000 | 2019-12-03 |
| 3 | Паразиты | | 2019-05-21 |

index

```
> select *  
    from film  
    where premiere > '2019-12-01';
```

index

```
> select *  
    from film  
    where premiere > '2019-12-01';
```

Какова сложность этого запроса - $O(?)$

index

```
> select *  
    from film  
    where premiere > '2019-12-01';
```

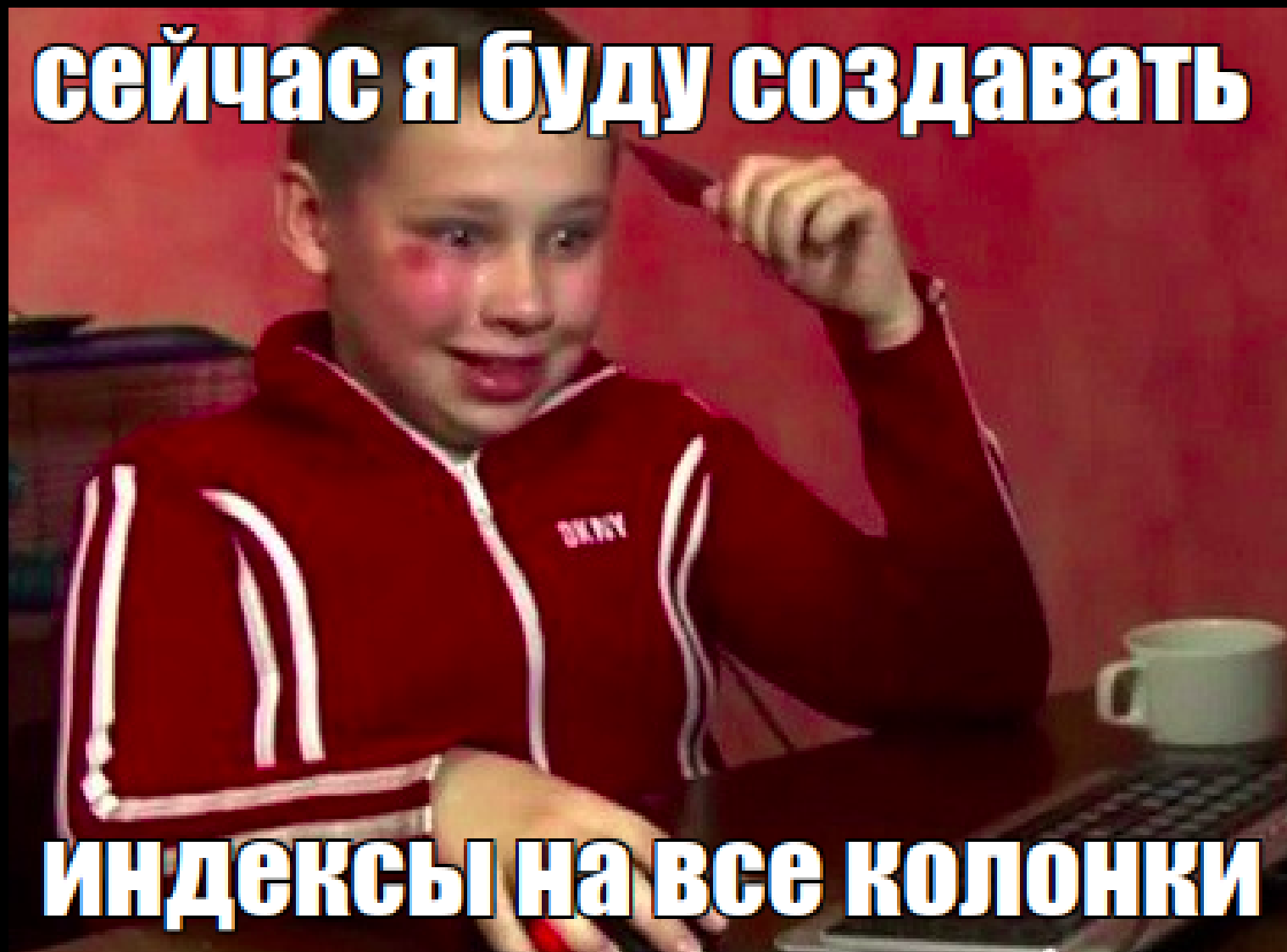
Какова сложность этого запроса - $O(?)$

Для фильтрации придётся проверить
каждую строку таблицы.

index

```
> create index film_premiere_idx  
    on film using btree(premiere);
```

сейчас я буду создавать



индексы на все колонки

index

index

- + ускоряет поиск записей;

index

- **+** ускоряет поиск записей;
- **-** замедляет вставку, обновление и удаление записей;

index

- + ускоряет поиск записей;
- - замедляет вставку, обновление и удаление записей;
- - занимает место на диске.

index

- + ускоряет поиск записей;
- - замедляет вставку, обновление и удаление записей;
- - занимает место на диске.

Лишние индексы - плохо!

Constraint

```
create table "user" (  
    id serial primary key,  
    email varchar not null,  
    registered_at date default now()  
);
```

Constraint

```
create table "user" (  
    id serial primary key,  
    email varchar not null,  
    registered_at date default now()  
);
```

```
insert into "user" (email) values  
    ('foo@kek.ru'), ('bar@kek.ru');
```

```
> select * from "user";
```

| id | email | registered_at |
|----|------------|---------------|
| 1 | foo@kek.ru | 2020-03-07 |
| 2 | bar@kek.ru | 2020-03-07 |

```
> select * from "user";
```

| id | email | registered_at |
|----|------------|---------------|
| 1 | foo@kek.ru | 2020-03-07 |
| 2 | bar@kek.ru | 2020-03-07 |

```
> insert into "user" (email)  
    values ( 'bar@kek.ru' );
```



```
> select * from "user";
```

| id | email | registered_at |
|----|------------|---------------|
| 1 | foo@kek.ru | 2020-03-07 |
| 2 | bar@kek.ru | 2020-03-07 |

```
> insert into "user" (email)
    values ( 'bar@kek.ru' );
```

```
> select * from "user";
```

| id | email | registered_at |
|----|------------|---------------|
| 1 | foo@kek.ru | 2020-03-07 |
| 2 | bar@kek.ru | 2020-03-07 |
| 3 | bar@kek.ru | 2020-03-10 |

Два пользователя с одинаковыми email 🤩

unique key

Мы должны были создать таблицу по-другому:

```
create table "user" (  
    id serial primary key,  
    email varchar not null unique,  
    registered_at date default now()  
);
```

unique key

Мы должны были создать таблицу по-другому:

```
create table "user" (  
    id serial primary key,  
    email varchar not null unique,  
    registered_at date default now()  
);
```

unique key

Мы должны были создать таблицу по-другому:

```
create table "user" (  
    id serial primary key,  
    email varchar not null unique,  
    registered_at date default now()  
);
```

Или если таблица уже создана:

```
alter table "user" add unique(email);
```

unique key

```
> select * from "user";
```

| id | email | registered_at |
|----|------------|---------------|
| 1 | foo@kek.ru | 2020-03-07 |
| 2 | bar@kek.ru | 2020-03-07 |

```
> insert into "user" (email) values ('bar@kek.ru');
```

```
ERROR:  duplicate key value violates unique  
        constraint "user_email_key"
```

```
> select * from "user";
```

| id | email | registered_at |
|----|------------|---------------|
| 1 | foo@kek.ru | 2020-03-07 |
| 2 | bar@kek.ru | 2020-03-07 |

```
> select * from film;
```

| id | title | budget | premiere |
|----|--------------|----------|------------|
| 1 | Джентельмены | 22000000 | 2019-12-03 |
| 3 | Паразиты | | 2019-05-21 |

```
> select * from "user";
```

| id | email | registered_at |
|----|------------|---------------|
| 1 | foo@kek.ru | 2020-03-07 |
| 2 | bar@kek.ru | 2020-03-07 |

```
> select * from film;
```

| id | title | budget | premiere |
|----|--------------|----------|------------|
| 1 | Джентельмены | 22000000 | 2019-12-03 |
| 3 | Паразиты | | 2019-05-21 |

```
create table film_rating (  
    id serial primary key,  
    user_id integer not null,  
    film_id integer not null,  
    rating integer not null  
);
```

```
> insert into film_rating
    (user_id, film_id, rating)
    values (1, 1, 7), (9123, 143523, 4);
```

```
> select * from film_rating;
```

| id | user_id | film_id | rating |
|----|---------|---------|--------|
| 1 | 1 | 1 | 7 |
| 2 | 9123 | 143523 | 4 |


```
> insert into film_rating
    (user_id, film_id, rating)
    values (1, 1, 7), (9123, 143523, 4);
```

```
> select * from film_rating;
```

| id | user_id | film_id | rating |
|----|---------|---------|--------|
| 1 | 1 | 1 | 7 |
| 2 | 9123 | 143523 | 4 |

film_rating соединяет несуществующего пользователя с несуществующим фильмом 🤔

foreign key

Мы должны были создать таблицу по-другому:

```
create table film_rating (  
    id serial primary key,  
    user_id integer not null references "user"(id),  
    film_id integer not null references "film"(id),  
    rating integer not null  
);
```

foreign key

Мы должны были создать таблицу по-другому:

```
create table film_rating (  
    id serial primary key,  
    user_id integer not null references "user"(id),  
    film_id integer not null references "film"(id),  
    rating integer not null  
);
```

foreign key

Мы должны были создать таблицу по-другому:

```
create table film_rating (  
    id serial primary key,  
    user_id integer not null references "user"(id),  
    film_id integer not null references "film"(id),  
    rating integer not null  
);
```

Или если таблица уже создана:

```
alter table film_rating  
    add constraint film_rating_user_id_fkey  
    foreign key (user_id) references "user" (id);  
  
-- аналогично для film_id
```

foreign key

```
> insert into film_rating  
  (user_id, film_id, rating)  
  values (1, 1, 7), (9123, 143523, 4);
```

```
ERROR:  insert or update on table "film_rating"  
violates foreign key constraint  
"film_rating_user_id_fkey"
```

```
DETAIL:  Key (user_id)=(9123) is not present  
in table "user".
```

foreign key

```
> select * from "user" where id = 1;
```

| id | email | registered_at |
|----|------------|---------------|
| 1 | foo@kek.ru | 2020-03-07 |

```
> select * from film_rating where id = 1;
```

| id | user_id | film_id | rating |
|----|---------|---------|--------|
| 1 | 1 | 1 | 7 |

foreign key

```
> select * from "user" where id = 1;
```

| id | email | registered_at |
|----|------------|---------------|
| 1 | foo@kek.ru | 2020-03-07 |

```
> select * from film_rating where id = 1;
```

| id | user_id | film_id | rating |
|----|---------|---------|--------|
| 1 | 1 | 1 | 7 |

```
> delete from "user" where id = 1;
```

foreign key

```
> select * from "user" where id = 1;
```

| id | email | registered_at |
|----|------------|---------------|
| 1 | foo@kek.ru | 2020-03-07 |

```
> select * from film_rating where id = 1;
```

| id | user_id | film_id | rating |
|----|---------|---------|--------|
| 1 | 1 | 1 | 7 |

```
> delete from "user" where id = 1;
```

ERROR: update or delete on table "user" violates
foreign key constraint "film_rating_user_id_fkey"
on table "film_rating"

DETAIL: Key (id)=(1) is still referenced from
table "film_rating".

foreign key

Мы могли бы создать таблицу по-другому:

```
create table film_rating (  
    id serial primary key,  
    user_id integer not null references "user"(id)  
        on delete cascade,  
    film_id integer not null references "film"(id),  
    rating integer not null  
);
```

foreign key

Мы могли бы создать таблицу по-другому:

```
create table film_rating (  
    id serial primary key,  
    user_id integer not null references "user"(id)  
        on delete cascade,  
    film_id integer not null references "film"(id),  
    rating integer not null  
);
```

foreign key

Мы могли бы создать таблицу по-другому:

```
create table film_rating (  
    id serial primary key,  
    user_id integer not null references "user"(id)  
        on delete cascade,  
    film_id integer not null references "film"(id),  
    rating integer not null  
);
```

Или если таблица уже создана:

```
alter table film_rating  
    drop constraint film_rating_user_id_fkey,  
    add constraint film_rating_user_id_fkey  
        foreign key (user_id) references "user"(id)  
        on delete cascade;
```

foreign key



```
> select * from "user";
```

| id | email | registered_at |
|----|------------|---------------|
| 1 | foo@kek.ru | 2020-03-07 |
| 2 | bar@kek.ru | 2020-03-07 |

```
> select * from film;
```

| id | title | budget | premiere |
|----|--------------|----------|------------|
| 1 | Джентельмены | 22000000 | 2019-12-03 |
| 3 | Паразиты | | 2019-05-21 |

```
> select * from "user";
```

| id | email | registered_at |
|----|------------|---------------|
| 1 | foo@kek.ru | 2020-03-07 |
| 2 | bar@kek.ru | 2020-03-07 |

```
> select * from film;
```

| id | title | budget | premiere |
|----|--------------|----------|------------|
| 1 | Джентельмены | 22000000 | 2019-12-03 |
| 3 | Паразиты | | 2019-05-21 |

```
> insert into film_rating (user_id, film_id, rating)
values (2, 3, 14);
```

```
> select * from film_rating;
```

| id | user_id | film_id | rating |
|----|---------|---------|--------|
| 8 | 2 | 3 | 14 |

```
> select * from "user";
```

| id | email | registered_at |
|----|------------|---------------|
| 1 | foo@kek.ru | 2020-03-07 |
| 2 | bar@kek.ru | 2020-03-07 |

```
> select * from film;
```

| id | title | budget | premiere |
|----|--------------|----------|------------|
| 1 | Джентельмены | 22000000 | 2019-12-03 |
| 3 | Паразиты | | 2019-05-21 |

```
> insert into film_rating (user_id, film_id, rating)
  values (2, 3, 14);
```

```
> select * from film_rating;
```

| id | user_id | film_id | rating |
|----|---------|---------|--------|
| 8 | 2 | 3 | 14 |

14/10 - неужели настолько хороший фильм? 🤔

check constraint

Мы должны были создать таблицу по-другому:

```
create table film_rating (  
    id serial primary key,  
    user_id integer not null references "user"(id),  
    film_id integer not null references "film"(id),  
    rating integer not null  
        check (rating > 0 and rating < 11)  
);
```


check constraint

Мы должны были создать таблицу по-другому:

```
create table film_rating (  
    id serial primary key,  
    user_id integer not null references "user"(id),  
    film_id integer not null references "film"(id),  
    rating integer not null  
        check (rating > 0 and rating < 11)  
);
```

check constraint

Мы должны были создать таблицу по-другому:

```
create table film_rating (  
    id serial primary key,  
    user_id integer not null references "user"(id),  
    film_id integer not null references "film"(id),  
    rating integer not null  
    check (rating > 0 and rating < 11)  
);
```

Или если таблица уже создана:

```
alter table film_rating  
    add constraint film_rating_rating_check  
    check (rating > 0 and rating < 11);
```

check constraint

```
> insert into film_rating (user_id, film_id, rating)
    values (2, 3, 14);
```

```
ERROR:  new row for relation "film_rating" violates
        check constraint "film_rating_rating_check"
DETAIL:  Failing row contains (3, 2, 3, 14).
```

Что ещё может пойти не так?

```
create table film_rating (  
    id serial primary key,  
    user_id integer not null references "user"(id),  
    film_id integer not null references "film"(id),  
    rating integer not null  
        check (rating > 0 and rating < 11)  
);
```

```
insert into film_rating (user_id, film_id, rating)
  values (2, 3, 10),
         (2, 3, 10),
         (2, 3, 10);
```

```
insert into film_rating (user_id, film_id, rating)
  values (2, 3, 10),
         (2, 3, 10),
         (2, 3, 10);
```

```
> select * from film_rating;
```

| id | user_id | film_id | rating |
|----|---------|---------|--------|
| 3 | 2 | 3 | 10 |
| 4 | 2 | 3 | 10 |
| 5 | 2 | 3 | 10 |

```
insert into film_rating (user_id, film_id, rating)
  values (2, 3, 10),
         (2, 3, 10),
         (2, 3, 10);
```

```
> select * from film_rating;
```

| id | user_id | film_id | rating |
|----|---------|---------|--------|
| 3 | 2 | 3 | 10 |
| 4 | 2 | 3 | 10 |
| 5 | 2 | 3 | 10 |



Составной unique key

Мы должны были создать таблицу по-другому:

```
create table film_rating (  
    id serial primary key,  
    user_id integer not null references "user"(id),  
    film_id integer not null references "film"(id),  
    rating integer not null  
        check (rating > 0 and rating < 11),  
    unique (user_id, film_id)  
);
```


Составной unique key

Мы должны были создать таблицу по-другому:

```
create table film_rating (  
    id serial primary key,  
    user_id integer not null references "user"(id),  
    film_id integer not null references "film"(id),  
    rating integer not null  
        check (rating > 0 and rating < 11),  
    unique (user_id, film_id)  
);
```

Составной unique key

Мы должны были создать таблицу по-другому:

```
create table film_rating (  
    id serial primary key,  
    user_id integer not null references "user"(id),  
    film_id integer not null references "film"(id),  
    rating integer not null  
        check (rating > 0 and rating < 11),  
    unique (user_id, film_id)  
);
```

Или если таблица уже создана:

```
alter table film_rating  
    add constraint film_rating_user_id_film_id_key  
    unique (user_id, film_id);
```

Составной unique key

```
> insert into film_rating (user_id, film_id, rating)
    values (2, 3, 10),
           (2, 3, 10),
           (2, 3, 10);
```

```
ERROR:  duplicate key value violates unique
        constraint "user_id_film_id_uq"
DETAIL:  Key (user_id, film_id)=(2, 3)
        already exists.
```

Index & Constraint

Index & Constraint

- Индексы создаём только на те колонки, по которым часто выполняем фильтрацию;

Index & Constraint

- Индексы создаём только на те колонки, по которым часто выполняем фильтрацию;
- Unique key, foreign key, check constraint - создаём так, чтобы гарантировать максимальную корректность данных;

Index & Constraint

- Индексы создаём только на те колонки, по которым часто выполняем фильтрацию;
- Unique key, foreign key, check constraint - создаём так, чтобы гарантировать максимальную корректность данных;
- Primary key - создаём отдельную "техническую" колонку;

Index & Constraint

- Индексы создаём только на те колонки, по которым часто выполняем фильтрацию;
- Unique key, foreign key, check constraint - создаём так, чтобы гарантировать максимальную корректность данных;
- Primary key - создаём отдельную "техническую" колонку;
- Constraint на уровне базы не отменяют проверку данных на уровне приложения;
- Foreign key должен ссылаться на первичный ключ.

foreign key

```
> select * from film_rating;
```

| id | user_id | film_id | rating |
|----|---------|---------|--------|
| 1 | 1 | 1 | 7 |

foreign key

```
> select * from film_rating;  
id | user_id | film_id | rating  
---+-----+-----+-----  
1  |        1 |        1 |      7
```

```
> select * from film_rating;  
id | user_email | film_id | rating  
---+-----+-----+-----  
1  | foo@kek.ru |        1 |      7
```

foreign key

```
> select * from film_rating;  
id | user_id | film_id | rating  
---+-----+-----+-----  
1  |      1  |      1  |      7
```

```
> select * from film_rating;  
id | user_email | film_id | rating  
---+-----+-----+-----  
1  | foo@kek.ru |      1  |      7
```

- Теперь труднее изменить email;

foreign key

```
> select * from film_rating;  
id | user_id | film_id | rating  
---+-----+-----+-----  
1  |      1 |      1 |      7
```

```
> select * from film_rating;  
id | user_email | film_id | rating  
---+-----+-----+-----  
1  | foo@kek.ru |      1 |      7
```

- Теперь труднее изменить email;
- А что, если завтра мы начнём авторизовываться через facebook, и email станет необязательным?



A front end developer eats
alone because he doesn't know
how to join tables

```
> select * from "user";
```

| id | email | registered_at |
|----|------------|---------------|
| 1 | foo@kek.ru | 2020-03-08 |
| 2 | bar@kek.ru | 2020-03-08 |

```
> select * from film;
```

| id | title | budget | premiere |
|----|--------------|-----------|------------|
| 1 | Джентельмены | 22000000 | 2019-12-03 |
| 3 | Паразиты | | 2019-05-21 |
| 2 | 1917 | 101000000 | 2019-12-04 |

```
> select * from film_rating;
```

| id | user_id | film_id | rating |
|----|---------|---------|--------|
| 1 | 1 | 1 | 7 |
| 2 | 1 | 3 | 10 |
| 3 | 2 | 1 | 6 |

```
> select
    film.id, film.title,
    film_rating.film_id, film_rating.rating
from film
join film_rating
    on film.id = film_rating.film_id;
```

| id | title |
|----|--------------|
| 1 | Джентельмены |
| 3 | Паразиты |
| 2 | 1917 |

| id | film_id | rating |
|----|---------|--------|
| 1 | 1 | 7 |
| 2 | 3 | 10 |
| 3 | 1 | 6 |

```
> select
    film.id, film.title,
    film_rating.film_id, film_rating.rating
from film
join film_rating
on film.id = film_rating.film_id;
```


| id | title |
|----|--------------|
| 1 | Джентельмены |
| 3 | Паразиты |
| 2 | 1917 |

| id | film_id | rating |
|----|---------|--------|
| 1 | 1 | 7 |
| 2 | 3 | 10 |
| 3 | 1 | 6 |

```
> select
    film.id, film.title,
    film_rating.film_id, film_rating.rating
from film
join film_rating
on film.id = film_rating.film_id;
```

| id | title |
|----|--------------|
| 1 | Джентельмены |
| 3 | Паразиты |
| 2 | 1917 |

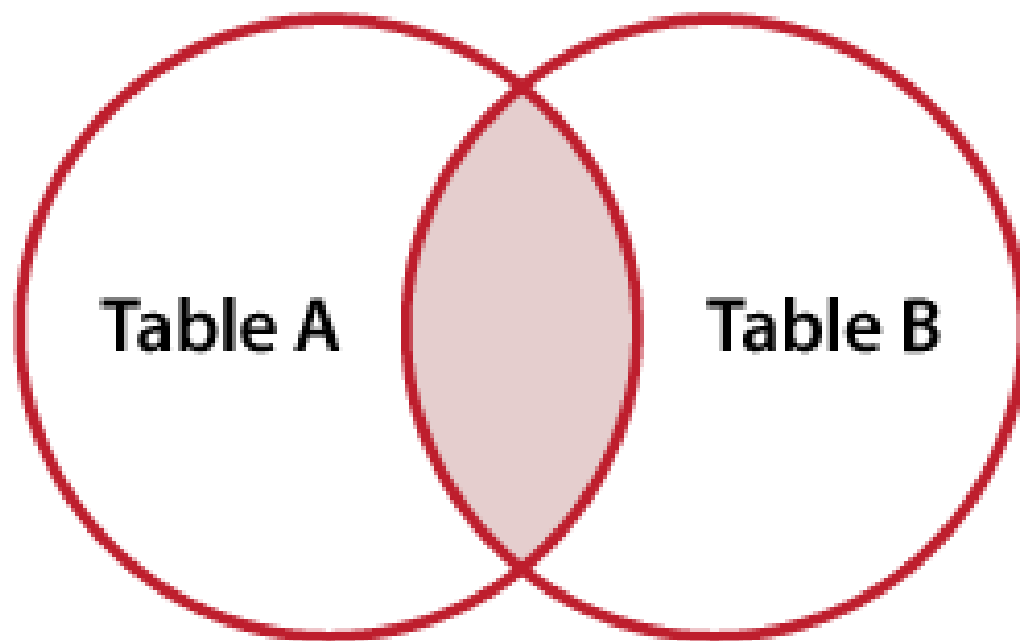
| id | film_id | rating |
|----|---------|--------|
| 1 | 1 | 7 |
| 2 | 3 | 10 |
| 3 | 1 | 6 |

```
> select
    film.id, film.title,
    film_rating.film_id, film_rating.rating
from film
join film_rating
on film.id = film_rating.film_id;
```

| id | title | film_id | rating |
|----|--------------|---------|--------|
| 1 | Джентельмены | 1 | 7 |
| 3 | Паразиты | 3 | 10 |
| 1 | Джентельмены | 1 | 6 |

| id | title | film_id | rating |
|----|--------------|---------|--------|
| 1 | Джентельмены | 1 | 7 |
| 3 | Паразиты | 3 | 10 |
| 1 | Джентельмены | 1 | 6 |

INNER JOIN



| id | title |
|----|--------------|
| 1 | Джентельмены |
| 3 | Паразиты |
| 2 | 1917 |

| id | film_id | rating |
|----|---------|--------|
| 1 | 1 | 7 |
| 2 | 3 | 10 |
| 3 | 1 | 6 |

```
> select
    film.id, film.title,
    film_rating.film_id, film_rating.rating
from film
left join film_rating
on film.id = film_rating.film_id;
```

| id | title | film_id | rating |
|----|--------------|---------|--------|
| 1 | Джентельмены | 1 | 7 |
| 3 | Паразиты | 3 | 10 |
| 1 | Джентельмены | 1 | 6 |
| 2 | 1917 | NULL | NULL |

| id | title |
|----|--------------|
| 1 | Джентельмены |
| 3 | Паразиты |
| 2 | 1917 |

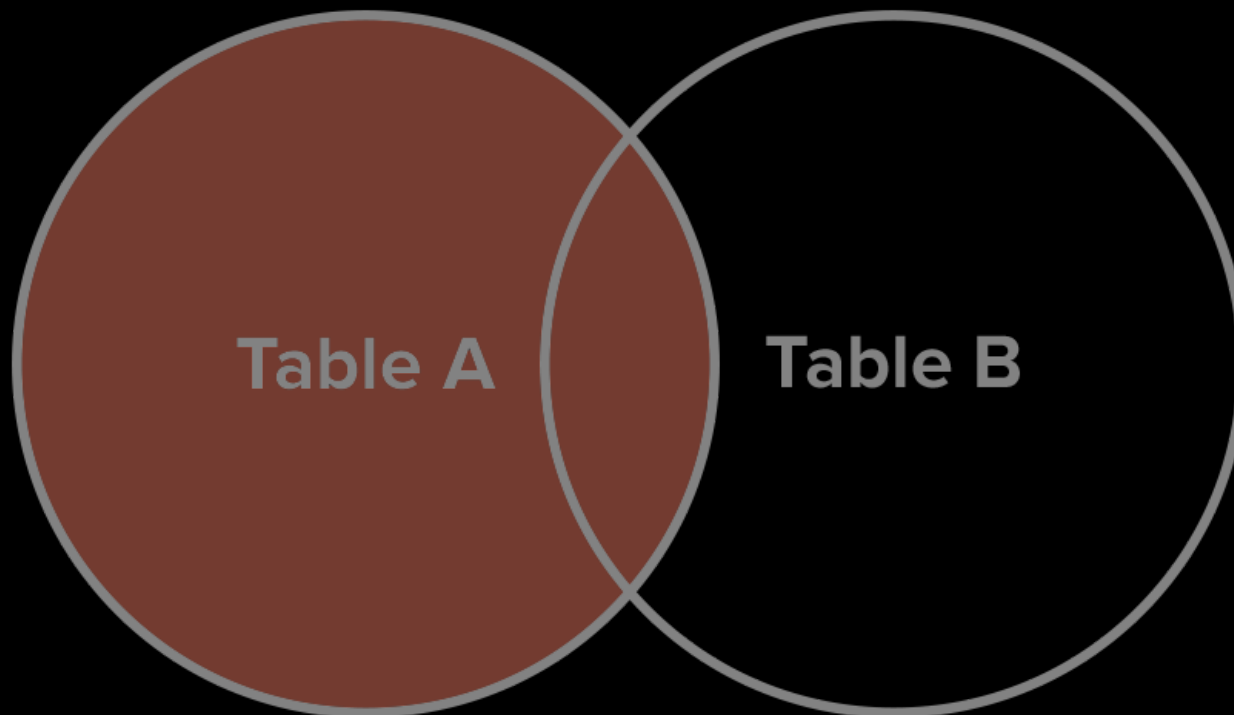
| id | film_id | rating |
|----|---------|--------|
| 1 | 1 | 7 |
| 2 | 3 | 10 |
| 3 | 1 | 6 |

```
> select
    film.id, film.title,
    film_rating.film_id, film_rating.rating
from film
    left join film_rating
        on film.id = film_rating.film_id;
```

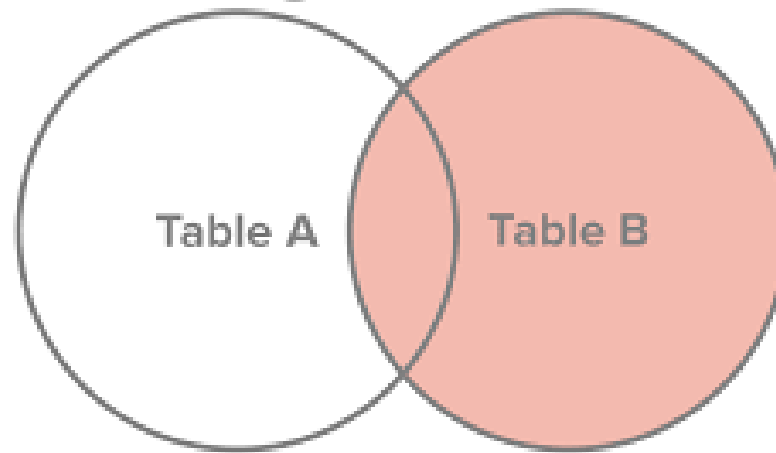
| id | title | film_id | rating |
|----|--------------|---------|--------|
| 1 | Джентельмены | 1 | 7 |
| 3 | Паразиты | 3 | 10 |
| 1 | Джентельмены | 1 | 6 |
| 2 | 1917 | NULL | NULL |

| id | title | film_id | rating |
|----|--------------|---------|--------|
| 1 | Джентельмены | 1 | 7 |
| 3 | Паразиты | 3 | 10 |
| 1 | Джентельмены | 1 | 6 |
| 2 | 1917 | NULL | NULL |

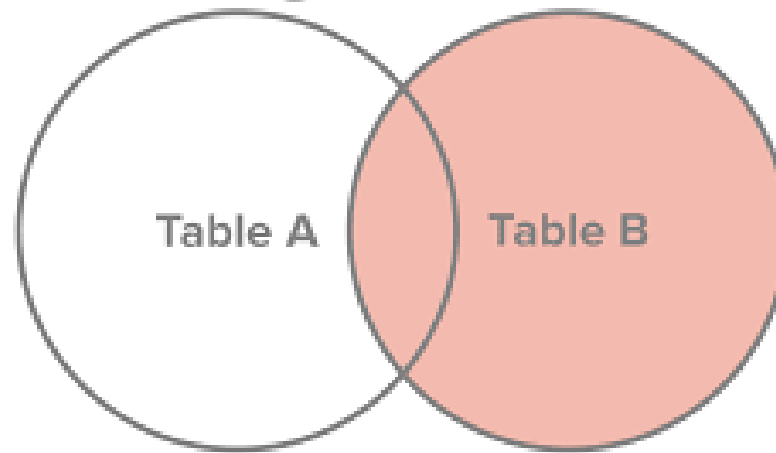
Left Join



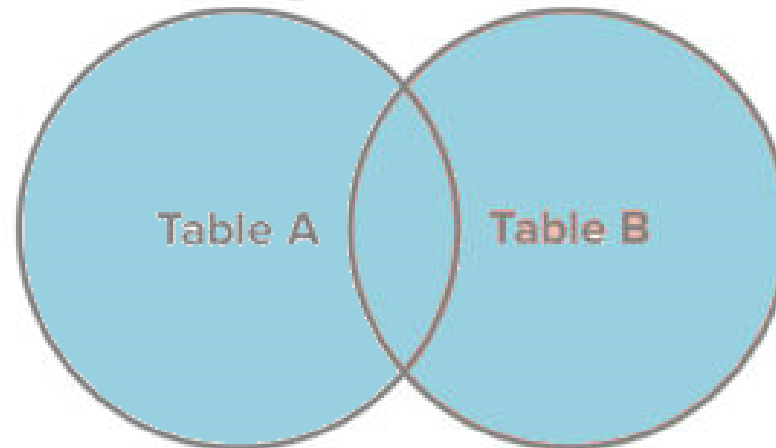
Right Join



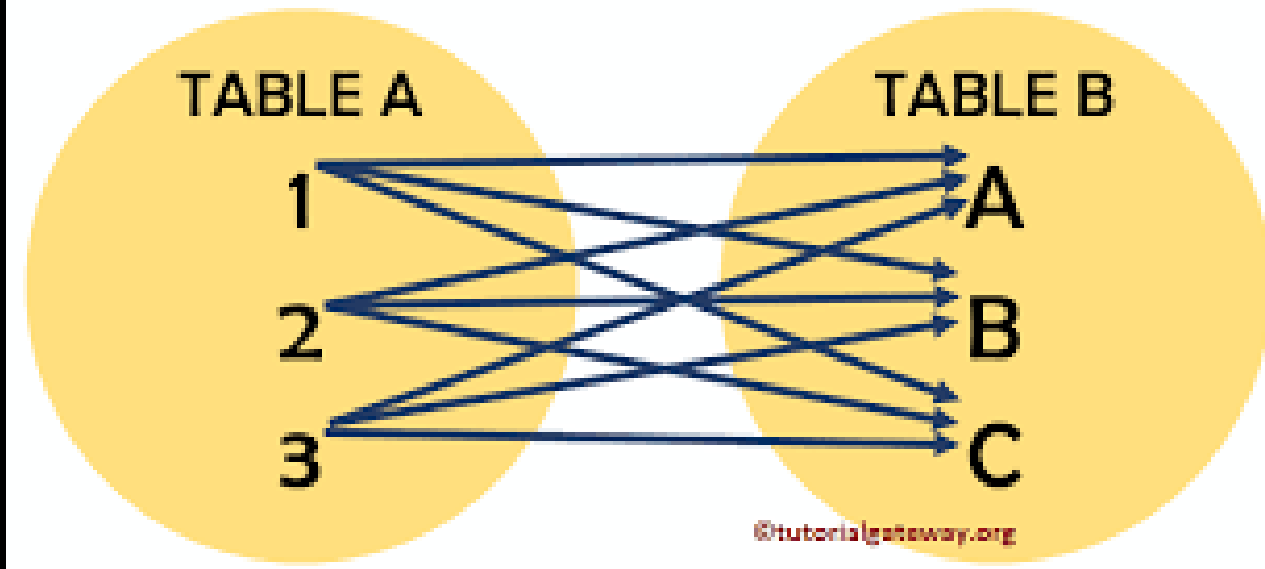
Right Join



Full Join

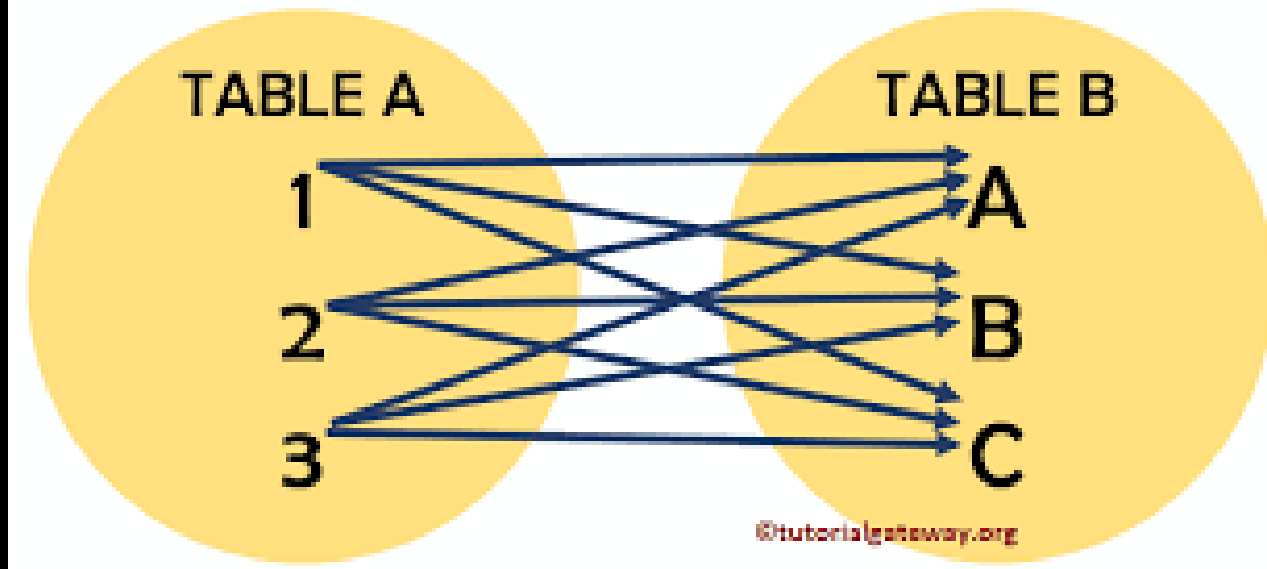


CROSS JOIN



```
> select film.id, film.title,  
        film_rating.film_id,  
        film_rating.rating  
  from film cross join film_rating;
```

CROSS JOIN



```
> select film.id, film.title,  
        film_rating.film_id,  
        film_rating.rating  
from film cross join film_rating;
```

```
> select film.id, film.title,  
        film_rating.film_id,  
        film_rating.rating  
  from film cross join film_rating;
```

| id | title | film_id | rating |
|----|--------------|---------|--------|
| 1 | Джентельмены | 1 | 7 |
| 3 | Паразиты | 1 | 7 |
| 2 | 1917 | 1 | 7 |
| 1 | Джентельмены | 3 | 10 |
| 3 | Паразиты | 3 | 10 |
| 2 | 1917 | 3 | 10 |
| 1 | Джентельмены | 1 | 6 |
| 3 | Паразиты | 1 | 6 |
| 2 | 1917 | 1 | 6 |

Как найти фильмы без оценок?

Как найти фильмы без оценок?

```
> select
    film.id, film.title,
    film_rating.film_id,
    film_rating.rating
from film
left join film_rating
on film.id = film_rating.film_id;
```

| id | title | film_id | rating |
|----|--------------|---------|--------|
| 1 | Джентельмены | 1 | 7 |
| 3 | Паразиты | 3 | 10 |
| 1 | Джентельмены | 1 | 6 |
| 2 | 1917 | NULL | NULL |

Как найти фильмы без оценок?

```
> select
    film.id, film.title,
    film_rating.film_id,
    film_rating.rating
from film
left join film_rating
on film.id = film_rating.film_id;
```

| id | title | film_id | rating |
|----|--------------|---------|--------|
| 1 | Джентельмены | 1 | 7 |
| 3 | Паразиты | 3 | 10 |
| 1 | Джентельмены | 1 | 6 |
| 2 | 1917 | NULL | NULL |

Как найти фильмы без оценок?

```
> select
    film.id, film.title,
    film_rating.film_id,
    film_rating.rating
from film as f
    left join film_rating as fr
        on f.id = fr.film_id
where fr.id is null;
```

| id | title | film_id | rating |
|----|-------|---------|--------|
| 2 | 1917 | NULL | NULL |

Как найти фильмы без оценок?

```
> select
    film.id, film.title,
    film_rating.film_id,
    film_rating.rating
from film as f
    left join film_rating as fr
        on f.id = fr.film_id
where fr.id is null;
```

| id | title | film_id | rating |
|----|-------|---------|--------|
| 2 | 1917 | NULL | NULL |

В запросе может быть много join

```
> select u.email, f.title, fr.rating
   from "user" u
      join film_rating fr
        on u.id = fr.user_id
      join film f
        on fr.film_id = f.id;
```

| email | title | rating |
|------------|--------------|--------|
| foo@kek.ru | Джентельмены | 7 |
| foo@kek.ru | Паразиты | 10 |
| bar@kek.ru | Джентельмены | 6 |

```
> select u.email, f.title, fr.rating  
      from "user" u  
        join film_rating fr  
          on u.id = fr.user_id  
        join film f  
          on fr.film_id = f.id;
```

Какова сложность этого запроса - $O(?)$

A man in a dark polo shirt and jeans stands next to a wooden barrel. Water is pouring out of the barrel. The background shows a wooden interior with shelves.

joін без индекса

A large concrete dam with water cascading over it, creating a massive waterfall. The dam is surrounded by lush green trees and a winding road.

joін с индексом

SQL - декларативный язык

Мы говорим, что хотим сделать, не как это сделать.

SQL - декларативный

ЯЗЫК

```
> select u.email, f.title, fr.rating
   from "user" u
      join film_rating fr
        on u.id = fr.user_id
      join film f
        on fr.film_id = f.id
  where
    f.premiere > '2019-12-01';
```

SQL - декларативный

ЯЗЫК

```
> select u.email, f.title, fr.rating
   from "user" u
      join film_rating fr
        on u.id = fr.user_id
      join film f
        on fr.film_id = f.id
  where
    f.premiere > '2019-12-01';
```

- Сначала приджойнить film_rating к user?

SQL - декларативный

ЯЗЫК

```
> select u.email, f.title, fr.rating
   from "user" u
      join film_rating fr
        on u.id = fr.user_id
      join film f
        on fr.film_id = f.id
  where
    f.premiere > '2019-12-01';
```

- Сначала приджойнить film_rating к user?
- Или сначала приджойнить film_rating к film?

SQL - декларативный

ЯЗЫК

```
> select u.email, f.title, fr.rating
   from "user" u
      join film_rating fr
        on u.id = fr.user_id
      join film f
        on fr.film_id = f.id
  where
    f.premiere > '2019-12-01';
```

- Сначала приджойнить film_rating к user?
- Или сначала приджойнить film_rating к film?
- Или сначала отфильтровать film по premiere?

SQL - декларативный

Порядок выполнения запроса зависит от
ЯЗЫК

SQL - декларативный

язык
Порядок выполнения запроса зависит от

- количества записей в таблицах;

SQL - декларативный

язык
Порядок выполнения запроса зависит от

- количества записей в таблицах;
- наличия индексов;

SQL - декларативный

язык
Порядок выполнения запроса зависит от

- количества записей в таблицах;
- наличия индексов;
- внутренних эвристик планировщика запросов.

SQL - декларативный

язык
Порядок выполнения запроса зависит от

- количества записей в таблицах;
- наличия индексов;
- внутренних эвристик планировщика запросов.

Как именно выполнится запрос? - оператор
"explain".

SQL - декларативный

Порядок выполнения запроса зависит от
ЯЗЫК

- количества записей в таблицах;
- наличия индексов;
- внутренних эвристик планировщика запросов.

Как именно выполнится запрос? - оператор
"explain".



Базы данных - зачем?

ACID в схемах и мемах.

A - атомарность (atomic)

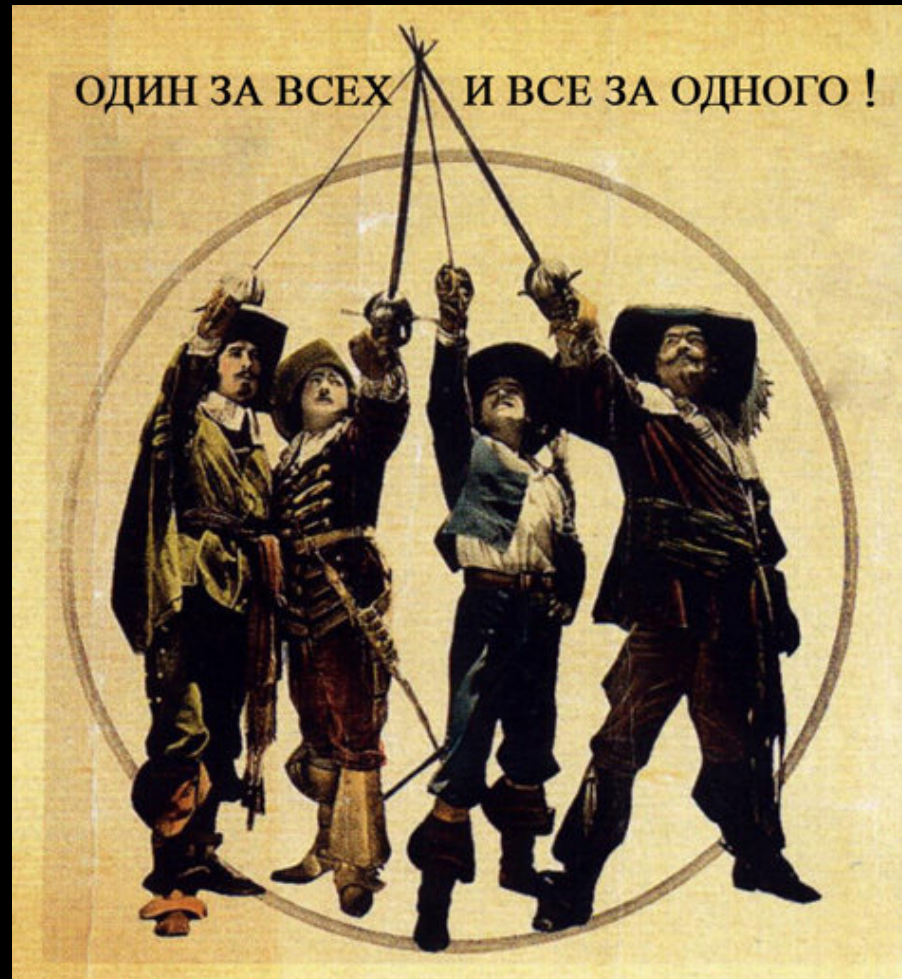
C - согласованность
(consistency)

I - изоляция (isolated)

D - долговечность (durability)

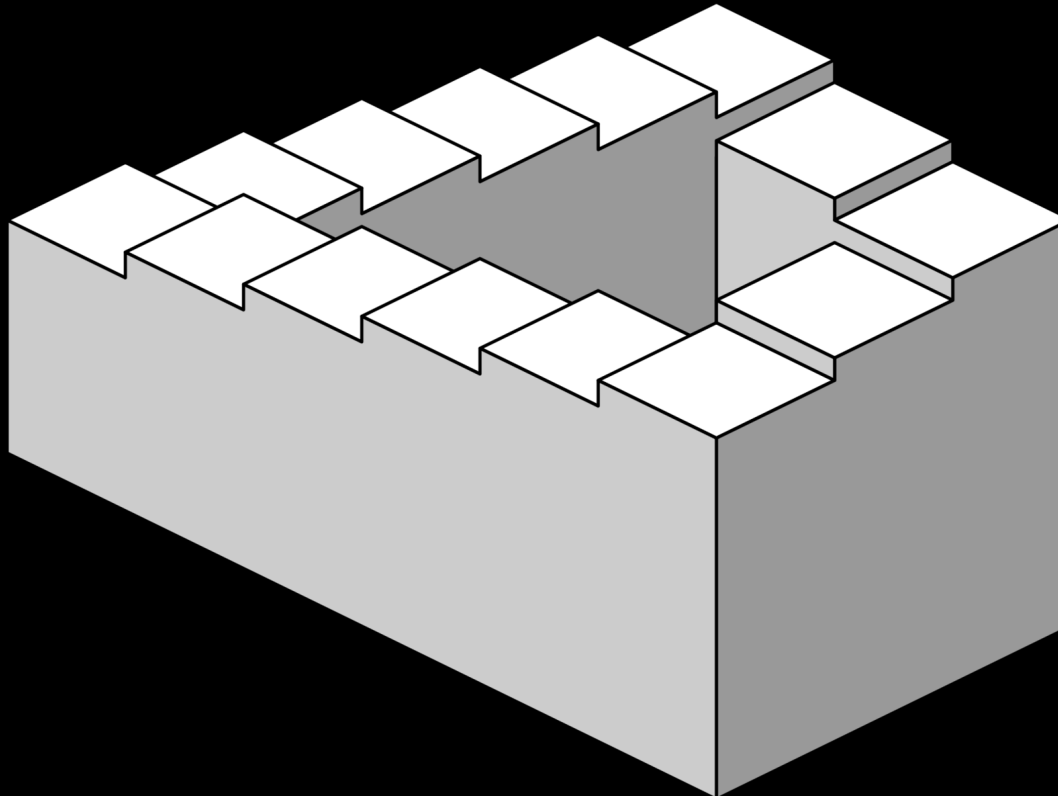
Атомарность (atomic)

Транзакция - набор операций, из которых либо должны быть выполнены все, либо не должна быть выполнена ни одна.



Согласованность (consistency)

В базе данных отсутствуют противоречия
с точки зрения приложения.



Изоляция (isolated)

Конкурирующие за доступ к БД транзакции физически обрабатываются последовательно и изолированно друг от друга, но для приложения это выглядит, как будто они выполняются параллельно.



Долговечность (durability)

Если транзакция завершилась успешно, то изменения будут доступны даже при аппаратном сбое.



УРОВНИ ИЗОЛЯЦИИ

- **Read uncommitted** (Чтение незафиксированных данных) - отсутствует в postgres;
- **Read committed** (Чтение зафиксированных данных);
- **Repeatable read** (Повторяемое чтение);
- **Serializable** (Сериализуемость).

Уровни изоляции транзакций

Уровни изоляции транзакций

- помним, что в базу может писать не один процесс;

Уровни изоляции транзакций

- помним, что в базу может писать не один процесс;
- выбираем уровень изоляции, исходя из специфики приложения;

Уровни изоляции транзакций

- помним, что в базу может писать не один процесс;
- выбираем уровень изоляции, исходя из специфики приложения;
- минимизируем локи - делаем транзакции как можно менее протяжёнными.

Нормализация

Набор рекомендаций по проектированию БД.

Нормализация

Набор рекомендаций по проектированию БД.

- исключение избыточности - когда данные хранятся в одном месте, их проще изменять;

Нормализация

Набор рекомендаций по проектированию БД.

- исключение избыточности - когда данные хранятся в одном месте, их проще изменять;
- обеспечение целостности - данные должны быть консистенты;

Нормализация

Набор рекомендаций по проектированию БД.

- исключение избыточности – когда данные хранятся в одном месте, их проще изменять;
- обеспечение целостности – данные должны быть консистенты;
- не стремимся уменьшить объём данных;

Нормализация

Набор рекомендаций по проектированию БД.

- исключение избыточности – когда данные хранятся в одном месте, их проще изменять;
- обеспечение целостности – данные должны быть консистенты;
 - не стремимся уменьшить объём данных;
 - не стремимся ускорить работу базы.

Первая нормальная форма (НФ1)

Первая нормальная форма (НФ1)

| user_id | watched_films |
|---------|---------------|
| 1 | 12,15,21 |
| 2 | 14,22 |

Первая нормальная форма (НФ1)

- значения в колонке должны быть атомарны;

| user_id | watched_films |
|---------|---------------|
| 1 | 12,15,21 |
| 2 | 14,22 |

Первая нормальная форма (НФ1)

- значения в колонке должны быть атомарны;

| user_id | watched_films |
|---------|---------------|
| 1 | 12,15,21 |
| 2 | 14,22 |

| user_id | film1 | film2 | film3 |
|---------|-------|-------|-------|
| 1 | 12 | 15 | 21 |
| 2 | 14 | 22 | |

Первая нормальная форма (НФ1)

- значения в колонке должны быть атомарны;
- колонки не должны дублировать друг друга;

| user_id | watched_films |
|---------|---------------|
| 1 | 12,15,21 |
| 2 | 14,22 |

| user_id | film1 | film2 | film3 |
|---------|-------|-------|-------|
| 1 | 12 | 15 | 21 |
| 2 | 14 | 22 | |

Первая нормальная форма (НФ1)

- значения в колонке должны быть атомарны;
- колонки не должны дублировать друг друга;
- каждая запись - представление одной сущности.

| user_id | watched_films |
|---------|---------------|
| 1 | 12,15,21 |
| 2 | 14,22 |

| user_id | film1 | film2 | film3 |
|---------|-------|-------|-------|
| 1 | 12 | 15 | 21 |
| 2 | 14 | 22 | |

НФ1

Каждая запись - представление одной сущности.

| user_id | film_id | date |
|---------|---------|------------|
| 1 | 12 | 2019-12-03 |
| 1 | 15 | 2019-10-02 |
| 1 | 21 | 2018-12-21 |
| 2 | 14 | 2020-02-12 |
| 2 | 22 | 2019-12-25 |

НФ1

Каждая запись - представление одной сущности.

| user_id | film_id | date |
|---------|---------|------------|
| 1 | 12 | 2019-12-03 |
| 1 | 15 | 2019-10-02 |
| 1 | 21 | 2018-12-21 |
| 2 | 14 | 2020-02-12 |
| 2 | 22 | 2019-12-25 |

Порядок записей не должен иметь значения.

Важен порядок - заводим явную колонку.

Потенциальный ключ

уникальный и минимальный набор атрибутов, которые описывают хранимую сущность.

НФ2 + НФ3

Храним в таблице только данные, которые зависят только от потенциального ключа целиком.

НФ2 + НФ3

Храним в таблице только данные, которые зависят только от потенциального ключа целиком.

Назначение задач

| user_id | task_id | task_project | project_state |
|---------|---------|--------------|---------------|
| 1 | 1 | talkbot | alpha |
| 1 | 2 | talkbot | alpha |
| 1 | 3 | chatbot | a b |
| 2 | 4 | chatbot | a b |
| 2 | 5 | chatbot | a b |

НФ2 + НФ3

Храним в таблице только данные, которые зависят только от потенциального ключа целиком.

Потенциальный ключ

Назначение задач

| user_id | task_id | task_project | project_state |
|---------|---------|--------------|---------------|
| 1 | 1 | talkbot | alpha |
| 1 | 2 | talkbot | alpha |
| 1 | 3 | chatbot | a b |
| 2 | 4 | chatbot | a b |
| 2 | 5 | chatbot | a b |

НФ2 + НФ3

Храним в таблице только данные, которые зависят только от потенциального ключа целиком.

Потенциальный ключ

Назначение задач

| user_id | task_id | task_project | project_state |
|---------|---------|--------------|---------------|
| 1 | 1 | talkbot | alpha |
| 1 | 2 | talkbot | alpha |
| 1 | 3 | chatbot | a b |
| 2 | 4 | chatbot | a b |
| 2 | 5 | chatbot | a b |

Проект зависит только от задачи, но не от пользователя.

НФ2 + НФ3

Назначение задач

| user_id | task_id |
|---------|---------|
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 2 | 4 |
| 2 | 5 |

$H\Phi 2 + H\Phi 3$

НФ2 + НФ3

Задачи

| number | title | deadline | project | project_state |
|--------|-------|------------|---------|---------------|
| 1872 | ... | 2020-03-16 | talkbot | alpha |
| 2012 | ... | 2020-03-16 | talkbot | alpha |
| 1923 | ... | 2020-03-17 | chatbot | a b |
| 069 | ... | 2020-03-16 | chatbot | a b |
| 1966 | ... | 2020-03-19 | chatbot | a b |

НФ2 + НФ3

Задачи

| number | title | deadline | project | project_state |
|--------|-------|------------|---------|---------------|
| 1872 | ... | 2020-03-16 | talkbot | alpha |
| 2012 | ... | 2020-03-16 | talkbot | alpha |
| 1923 | ... | 2020-03-17 | chatbot | a b |
| 069 | ... | 2020-03-16 | chatbot | a b |
| 1966 | ... | 2020-03-19 | chatbot | a b |

НФ2 + НФ3

Задачи

| number | title | deadline | project | project_state |
|--------|-------|------------|---------|---------------|
| 1872 | ... | 2020-03-16 | talkbot | alpha |
| 2012 | ... | 2020-03-16 | talkbot | alpha |
| 1923 | ... | 2020-03-17 | chatbot | a b |
| 069 | ... | 2020-03-16 | chatbot | a b |
| 1966 | ... | 2020-03-19 | chatbot | a b |

Состояние проекта зависит не от потенциального ключа, а от колонки "проект".

НФ2 + НФ3

Задачи

| number | title | deadline | project_id |
|--------|-------|------------|------------|
| 1872 | ... | 2020-03-16 | 1 |
| 2012 | ... | 2020-03-16 | 1 |
| 1923 | ... | 2020-03-17 | 2 |
| 069 | ... | 2020-03-16 | 2 |
| 1966 | ... | 2020-03-19 | 2 |

НФ2 + НФ3

Задачи

| number | title | deadline | project_id |
|--------|-------|------------|------------|
| 1872 | ... | 2020-03-16 | 1 |
| 2012 | ... | 2020-03-16 | 1 |
| 1923 | ... | 2020-03-17 | 2 |
| 069 | ... | 2020-03-16 | 2 |
| 1966 | ... | 2020-03-19 | 2 |

Проекты

| project | state |
|---------|-------|
| talkbot | alpha |
| chatbot | a b |

Денормализация

Отклонения от нормальных форм ради удобства или скорости работы.

| id | title | aver_rating |
|----|--------------|-------------|
| 1 | Джентельмены | 8.70 |
| 3 | Паразиты | 8.06 |
| 2 | 1917 | 8.11 |

| id | user_id | film_id | rating |
|-----|---------|---------|--------|
| 1 | 1 | 1 | 7 |
| 2 | 1 | 3 | 10 |
| ... | ... | ... | ... |

Денормализация

Отклонения от нормальных форм ради удобства или скорости работы.

| id | title | aver_rating |
|----|--------------|-------------|
| 1 | Джентельмены | 8.70 |
| 3 | Паразиты | 8.06 |
| 2 | 1917 | 8.11 |

| id | user_id | film_id | rating |
|-----|---------|---------|--------|
| 1 | 1 | 1 | 7 |
| 2 | 1 | 3 | 10 |
| ... | ... | ... | ... |

- считать средний рейтинг фильма долго;

Денормализация

Отклонения от нормальных форм ради удобства или скорости работы.

| id | title | aver_rating |
|----|--------------|-------------|
| 1 | Джентельмены | 8.70 |
| 3 | Паразиты | 8.06 |
| 2 | 1917 | 8.11 |

| id | user_id | film_id | rating |
|-----|---------|---------|--------|
| 1 | 1 | 1 | 7 |
| 2 | 1 | 3 | 10 |
| ... | ... | ... | ... |

- считать средний рейтинг фильма долго;
- супер-точное значение рейтинга не требуется.

Нормализация

Нормализация

- набор *рекомендаций* по проектированию БД;

Нормализация

- набор *рекомендаций* по проектированию БД;
- сначала нормализация, потом денормализация.

Перерыв?

SQLite

база данных в одном файле

SQLite

база данных в одном файле

+ не требует усилий по развёртыванию и настройке;

SQLite

база данных в одном файле

- + не требует усилий по развёртыванию и настройке;
- + поддержка в `python` из коробки;

SQLite

база данных в одном файле

- + не требует усилий по развёртыванию и настройке;
- + поддержка в `python` из коробки;
- нельзя одновременно писать из двух процессов;

SQLite

база данных в одном файле

- + не требует усилий по развёртыванию и настройке;
- + поддержка в `python` из коробки;
- нельзя одновременно писать из двух процессов;
- нельзя работать по сети.

Серверные базы данных

Серверные базы данных

- Oracle;

Серверные базы данных

- Oracle;
- Microsoft SQL Server;

Серверные базы данных

- Oracle;
- Microsoft SQL Server;
- MySQL;

Серверные базы данных

- Oracle;
- Microsoft SQL Server;
- MySQL;
- Postgres;
- etc...

Что делаем из python?

Вариант 1: работаем с "сырым" sql через драйвер.

raw sql

```
1 import sqlite3
2
3 conn = sqlite3.connect('example.db')
4 c = conn.cursor()
```

raw sql

```
1 import sqlite3
2
3 conn = sqlite3.connect('example.db')
4 c = conn.cursor()
```

```
1 c.execute("""
2     CREATE TABLE film (
3         ID INTEGER PRIMARY KEY,
4         title varchar, budget integer, premiere date
5     )
6 """)
7
8 c.execute("""
9     INSERT INTO film (title, budget, premiere) VALUES
10         ('Джентельмены', 22000000, '2019-12-03'),
11         ('1917', 100000000, '2019-12-04')
12 """)
```

raw sql

```
1 import sqlite3
2
3 conn = sqlite3.connect('example.db')
4 c = conn.cursor()
```

```
1 c.execute("""
2     CREATE TABLE film (
3         ID INTEGER PRIMARY KEY,
4         title varchar, budget integer, premiere date
5     )
6 """)
7
8 c.execute("""
9     INSERT INTO film (title, budget, premiere) VALUES
10         ('Джентельмены', 22000000, '2019-12-03'),
11         ('1917', 100000000, '2019-12-04')
12 """)
```

Нет подсветки sql-синтаксиса* 🙄

raw sql

```
1 res = c.execute("SELECT * FROM film;").fetchall()
2 for row in res:
3     print(row)
4
5 # (1, 'Джентельмены', 22000000, '2019-12-03')
6 # (2, '1917', 100000000, '2019-12-04')
7
8 conn.commit()
9 conn.close()
```

raw sql

sql-инъекции

```
film_id = 1
```

```
# ПЛОХО
```

```
cursor.execute(f"""  
    SELECT * FROM film WHERE id = {film_id}  
""")
```



```
1 film_id = '1; drop table film;'
2
3 # ПЛОХО
4 cursor.execute(f"""
5     SELECT * FROM film WHERE id = {film_id}
6 """)
7
8 # SELECT * FROM film
9 # WHERE id = 1; drop table film;
```



```
1 film_id = '1; drop table film;'
2
3 # ПЛОХО
4 cursor.execute(f"""
5     SELECT * FROM film WHERE id = {film_id}
6 """)
7
8 # SELECT * FROM film
9 # WHERE id = 1; drop table film;
```



sqlite эта инъекция не ломает, потому что драйвер не может выполнить больше одного запроса за раз.

raw sql

sql-инъекции

```
film_id = 1
```

```
# ПЛОХО
```

```
cursor.execute(f"""  
    SELECT * FROM film WHERE id = {film_id}  
""")
```

raw sql

sql-инъекции

```
film_id = 1
```

```
# плохо
```

```
cursor.execute(f"""  
    SELECT * FROM film WHERE id = {film_id}  
""")
```

```
# хорошо
```

```
cursor.execute("""  
    SELECT * FROM film WHERE id = ?  
""", user_id  
)
```

? - placeholder, который сообщает драйверу, что в этом месте будет параметр, который нужно экранировать.

```

@staticmethod
def _dashboard_sql_query():
    time_format = '%Y-%m-%d %H:%M:%S'
    date_format = '%Y-%m-%d'
    current_time = timezone.localtime()
    meta_db_name = connections['meta'].settings_dict['NAME']

    sql_raw = """
SELECT threadid,
       state,
       lol_id,
       visitor_name,
       first_visitor_message_time,
       kek_id,
       kek_fullname,
       kek_subdivision,
       lol_subdivision,
       waiting_time,
       activity_time,
       (CASE
            WHEN kek_id IS NOT NULL THEN
                SUBSTRING_INDEX(last_messages, '^'[^*, 3)
            ELSE
                last_messages
        END) last_messages
FROM
    (SELECT threads.threadid,
           threads.state,
           threads.departmentid lol_id,
           threads.kekid kek_id,
           threads.visitor_name,
           threads.first_visitor_message_time,
           keks.fullname kek_fullname,
           o_subdivision.subdivisionid kek_subdivision,
           c_department.departmentsubdivision lol_subdivision,
           TIME_TO_SEC(COALESCE(CAST(Timediff(threads.first_kek_message, threads.created) AS CHAR),
                                TIMEDIFF("{current_time.strftime(time_format)}", threads.first_visitor_messa
           TIME_TO_SEC(timediff('{current_time.strftime(time_format)}', last_kek_message.created)) activity_time,
           (CASE
                WHEN threads.kekid IS NOT NULL THEN
                    (SELECT GROUP_CONCAT(message SEPARATOR '^'[^*)
FROM chatmessage

```

raw sql

Много затрат на построение запроса.

```
1 def find_any(columns):
2     conditions = [
3         f"{column} = '?' "
4         for column in columns
5     ]
6     return ' OR \n'.join(conditions)
7
8 # field1 = '?' OR
9 # field2 = '?' OR
10 # field3 = '?'
```

raw sql

Берём драйвер и просто* исполняем
запросы.

raw sql

Берём драйвер и просто* исполняем запросы.

+ видим конечный sql-запрос;

raw sql

Берём драйвер и просто* исполняем запросы.

- + видим конечный sql-запрос;
- результаты запросов - простые типы;

raw sql

Берём драйвер и просто* исполняем запросы.

- + видим конечный sql-запрос;
- результаты запросов - простые типы;
- мучаемся с шаблонами;

raw sql

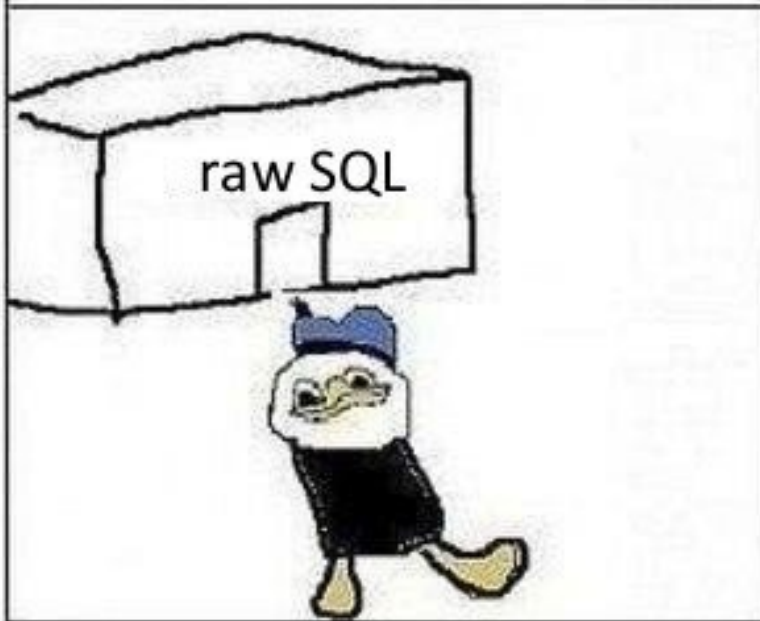
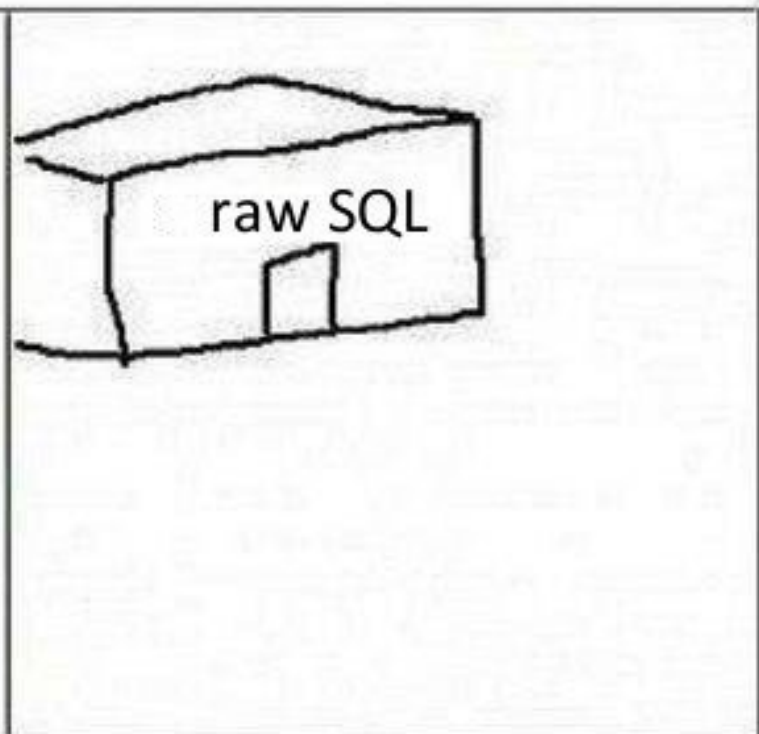
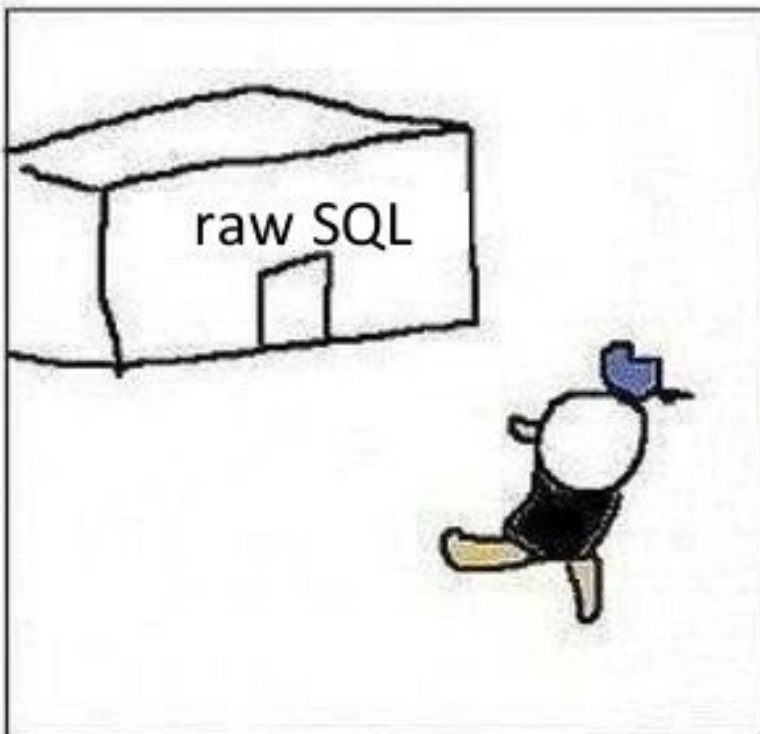
Берём драйвер и просто* исполняем запросы.

- + видим конечный sql-запрос;
- результаты запросов - простые типы;
- мучаемся с шаблонами;
- помним про инъекции;

raw sql

Берём драйвер и просто* исполняем запросы.

- + видим конечный sql-запрос;
- результаты запросов - простые типы;
- мучаемся с шаблонами;
- помним про инъекции;
- зависим от диалекта sql.



Что делаем из python?

Вариант 2: object relation mapping
(ORM)

A close-up photograph of a light-colored dog's face, focusing on its eyes and nose. Two small, orange-brown dog figurines are placed on a surface in front of the dog's nose. The text 'мапътес' is overlaid at the top, 'таблички' is on the left, and 'классы' is on the right.

мапътес

таблички

классы

ORM

ORM

- Работаем с базой в терминах ООП, а не в терминах реляционной алгебры.

ORM

- Работаем с базой в терминах ООП, а не в терминал реляционной алгебры.
- Есть `query build` - не форматируем строки вручную на `python`.

ORM

- Работаем с базой в терминах ООП, а не в терминал реляционной алгебры.
- Есть `query build` - не форматируем строки вручную на `python`.
- Не зависим от платформы*: сегодня `sqlite`, завтра `postgres`.

peewee

```
1 from peewee import (  
2     SqliteDatabase, Model, CharField,  
3     DateField, AutoField)  
4  
5 db = SqliteDatabase('people.db')  
6  
7  
8 class Film(Model):  
9     id = AutoField()  
10    title = CharField()  
11    premiere = DateField()  
12  
13    class Meta:  
14        database = db  
15  
16  
17 db.connect()  
18 db.create_tables([Film])
```

peewee

```
1 from datetime import date
2
3 film = Film(
4     title='Джентльмены',
5     premiere=date(2019, 12, 3))
6 film.save()
7
8 loaded_film = (
9     Film.select()
10    .where(Film.title == 'Джентльмены')
11    .get()
12 )
13 print(loaded_film.title)
14 # Джентльмены
```

django orm

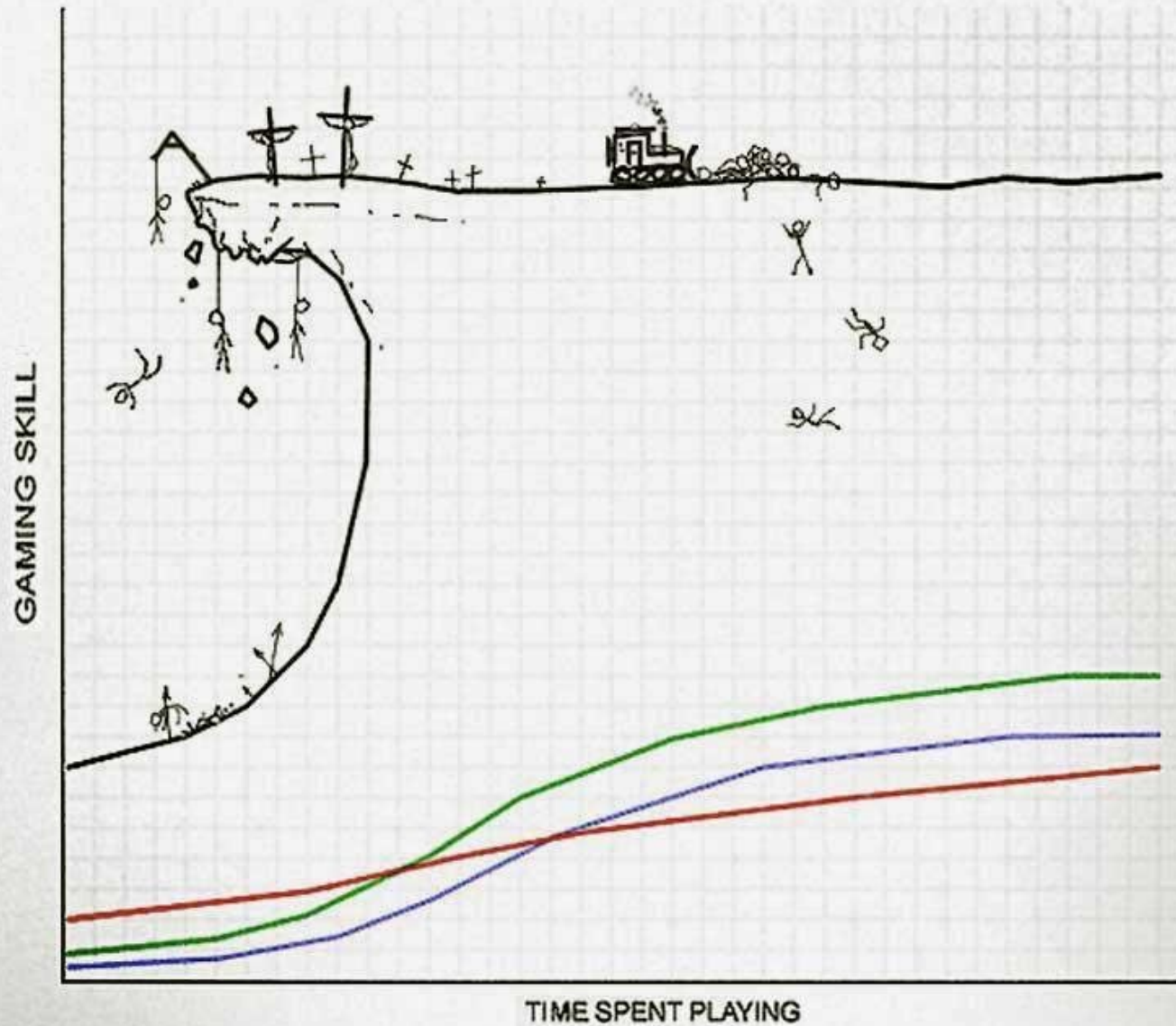
```
1 from django.db import models
2
3
4 class Film(models.Model):
5     title = models.CharField(max_length=100)
6     premiere = models.DateTimeField()
7
8     def __str__(self):
9         return self.title
```

django orm

```
1 b = Film(  
2     title='Джентельмены',  
3     premiere=date(2019, 12, 3))  
4 b.save()  
5  
6 loaded_film = Film.objects.get(title='Джентельмены')  
7 print(loaded_film)  
8 # Джентельмены
```

SQLAlchemy

LEARNING CURVES OF POPULAR MMORPGS



World of Warcraft
POTBS

LOTR Online

SQLAlchemy

SQLAlchemy

orm

```
1 from datetime import date
2 import sqlalchemy as sa
3 from sqlalchemy.ext.declarative import declarative_base
4
5 Base = declarative_base()
6
7
8 class Film(Base):
9     __tablename__ = 'film'
10
11     id = sa.Column(sa.Integer, primary_key=True)
12     title = sa.Column(sa.String())
13     premiere = sa.Column(sa.Date())
14
15
16 film = Film(
17     title='Джентельмены',
18     premiere=date(2019, 12, 3))
```

SQLAlchemy

engine - связь SQLAlchemy с базой

```
1 import sqlalchemy as sa
2
3 engine = sa.create_engine('sqlite:///foo.db')
4
5 # ИЛИ
6 # `pip install psycopg2`
7 db_url = 'postgresql://scott:tiger@localhost:5432/mydatabase'
8 engine = sa.create_engine(db_url)
9
10
11 Base.metadata.create_all(engine)
```

SQLAlchemy

session - "близнец" транзакции на уровне python

```
1 from sqlalchemy.orm import sessionmaker
2
3 Session = sessionmaker(bind=engine)
4 session = Session()
5
6 film = Film(
7     title='Джентельмены',
8     premiere=date(2019, 12, 3))
9 session.add(film) # insert
10
11 film.title = 'new title'
12 session.add(film) # update
13
14 session.delete(film) # delete
15
16 # КОММИТИМ ИЗМЕНЕНИЯ
17 session.commit()
18
19 # или откатываем изменения
20 session.rollback()
```

SQLAlchemy

session нужно закрыть после использования

```
1 from contextlib import contextmanager
2
3 Session = sessionmaker(bind=engine)
4
5
6 @contextmanager
7 def create_session(**kwargs):
8     new_session = Session(**kwargs)
9     try:
10         yield new_session
11         new_session.commit()
12     except Exception:
13         new_session.rollback()
14         raise
15     finally:
16         new_session.close()
```

SQLAlchemy

`session` - "близнец" транзакции на уровне python

```
1 with create_session() as session:
2     film = Film(
3         title='Джентельмены',
4         premiere=date(2019, 12, 3),
5     )
6
7     session.add(film)
```

SQLAlchemy

session.query

```
1 with create_session() as session:
2     test_user = (
3         session.query(Film)
4         .filter(Film.title == '1917')
5         .one()
6     )
7
8 # SELECT film.id AS film_id,
9 #         film.title AS film_title,
10 #        film.premiere AS film_premiere
11 # FROM film
12 # WHERE film.title = ?
```

SQLAlchemy

```
1 films = (  
2     session.query(Film)  
3     .filter(Film.title == 'Жизнь прекрасна')  
4     .one()  
5 )  
6  
7 # sqlalchemy.orm.exc.NoResultFound:  
8 #     No row was found for one()
```


SQLAlchemy

| метод | несколько строк | одна строка | ни одной строки |
|-------------|-----------------------|--------------------------|-----------------|
| all | список | список из одного объекта | пустой список |
| one | исключение | объект | исключение |
| first | случайный из объектов | объект | None |
| one_or_none | исключение | объект | None |

SQLAlchemy

```
1 with create_session() as session:
2     user = User(name='admin')
3     session.add(user)
4
5
6 with create_session() as session:
7     user = session.query(User).first()
8     print(user.name)
9
10 > 'admin'
```

SQLAlchemy

```
1 with create_session() as session:
2     user = session.query(User).first()
3
4     ... # что-то ещё
5 print(user.name)
6
7
8 > Traceback (most recent call last):
9 >     ...
10 > sqlalchemy.orm.exc.DetachedInstanceError: ...
```

SQLAlchemy

```
1 with create_session() as session:
2     user = session.query(User).first()
3
4     ... # что-то ещё
5 print(user.name)
6
7 > Traceback (most recent call last):
8 > ...
9 > sqlalchemy.orm.exc.DetachedInstanceError: ...
```



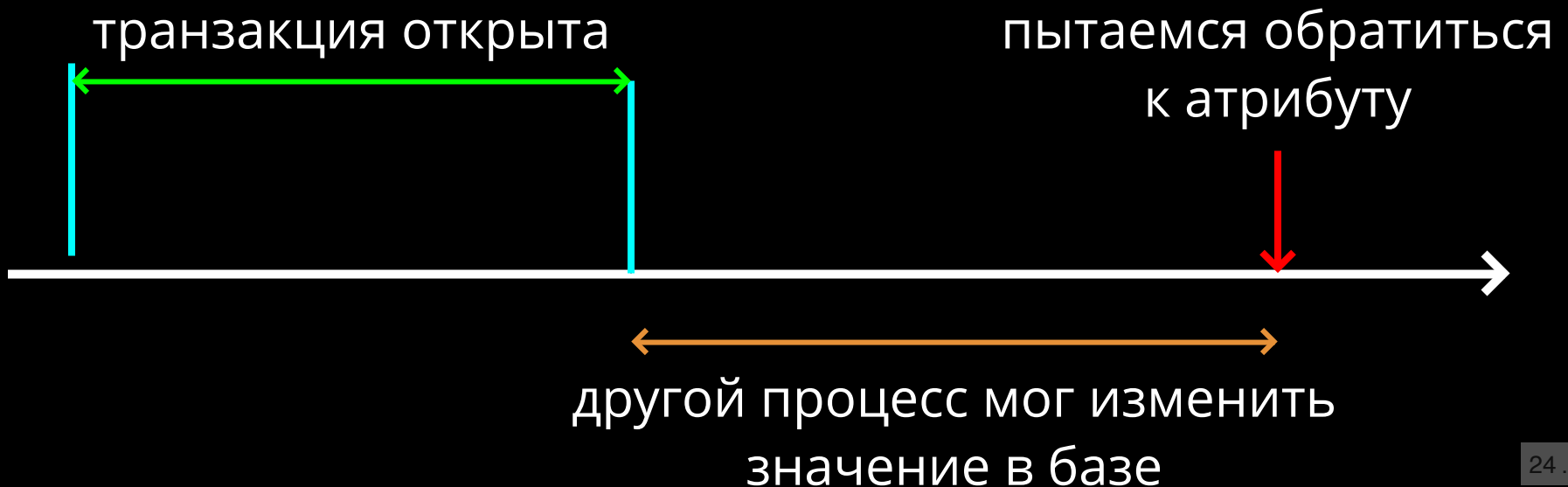
SQLAlchemy

```
1 with create_session() as session:
2     user = session.query(User).first()
3
4     ... # что-то ещё
5 print(user.name)
6
7 > Traceback (most recent call last):
8 > ...
9 > sqlalchemy.orm.exc.DetachedInstanceError: ...
```



SQLAlchemy

```
1 with create_session() as session:
2     user = session.query(User).first()
3
4     ... # что-то ещё
5 print(user.name)
6
7 > Traceback (most recent call last):
8 > ...
9 > sqlalchemy.orm.exc.DetachedInstanceError: ...
```



SQLAlchemy

```
1 with create_session() as session:  
2     user = session.query(User).first()  
3     name = user.name  
4  
5 print(name)  
6 > 'admin'
```

SQLAlchemy

```
1 with create_session() as session:  
2     user = session.query(User).first()  
3     name = user.name  
4  
5 print(name)  
6 > 'admin'
```

```
1 with create_session(expire_on_commit=False) as session:  
2     user = session.query(User).first()  
3  
4 print(user.name)  
5 > 'admin'
```


SQLAlchemy

```
1 with create_session() as session:
2     user = session.query(User).first()
3     name = user.name
4
5 print(name)
6 > 'admin'
```

```
1 with create_session(expire_on_commit=False) as session:
2     user = session.query(User).first()
3
4 print(user.name)
5 > 'admin'
```

expire_on_commit - когда мы уверены, что с базой работает только наш процесс, например, в тестах

flush

```
1 with create_session() as session:  
2     user = User(...)  
3     session.add(user)  
4     print(user.id)  
5 # None
```

flush

```
1 with create_session() as session:
2     user = User(...)
3     session.add(user)
4     print(user.id)
5 # None
```

```
1 with create_session() as session:
2     user = User(...)
3     session.add(user)
4     session.flush()
5     print(user.id)
6 # 1
```

flush

```
1 with create_session() as session:
2     user = User(...)
3     session.add(user)
4     print(user.id)
5 # None
```

```
1 with create_session() as session:
2     user = User(...)
3     session.add(user)
4     session.flush()
5     print(user.id)
6 # 1
```

flush - выполнить запрос, но не коммитить транзакцию.

SQLAlchemy

starter pack

```
1 engine = sa.create_engine(...)
2 Session = sessionmaker(bind=engine)
3 Base = declarative_base()
4
5
6 @contextmanager
7 def create_session(**kwargs):
8     ...
9
10
11 class User(Base):
12     __tablename__ = 'user'
13
14     ...
15
16
17 Base.metadata.create_all(engine)
18
19 with create_session() as session:
20     ...
```

SQLAlchemy

starter pack

```
1 engine = sa.create_engine(...)
2 Session = sessionmaker(bind=engine)
3 Base = declarative_base()
4
5
6 @contextmanager
7 def create_session(**kwargs):
8     ...
9
10
11 class User(Base):
12     __tablename__ = 'user'
13
14     ...
15
16
17 Base.metadata.create_all(engine)
18
19 with create_session() as session:
20     ...
```

- подключение к базе - engine;

SQLAlchemy

starter pack

```
1 engine = sa.create_engine(...)
2 Session = sessionmaker(bind=engine)
3 Base = declarative_base()
4
5
6 @contextmanager
7 def create_session(**kwargs):
8     ...
9
10
11 class User(Base):
12     __tablename__ = 'user'
13
14     ...
15
16
17 Base.metadata.create_all(engine)
18
19 with create_session() as session:
20     ...
```

- подключение к базе - engine;
- описываем таблицы - Base;

SQLAlchemy

starter pack

```
1 engine = sa.create_engine(...)
2 Session = sessionmaker(bind=engine)
3 Base = declarative_base()
4
5
6 @contextmanager
7 def create_session(**kwargs):
8     ...
9
10
11 class User(Base):
12     __tablename__ = 'user'
13
14     ...
15
16
17 Base.metadata.create_all(engine)
18
19 with create_session() as session:
20     ...
```

- подключение к базе - engine;
- описываем таблицы - Base;
- работаем с сессией через контекстный менеджер.

ОТНОШЕНИЯ

```
1 class UserRole(Base):
2     __tablename__ = 'user_role'
3
4     id = sa.Column(sa.Integer, primary_key=True)
5     name = sa.Column(sa.String, unique=True)
6     ...
7
8
9 class User(Base):
10     __tablename__ = 'user'
11
12     id = sa.Column(sa.Integer, primary_key=True)
13     email = sa.Column(sa.String, unique=True)
14     role_id = sa.Column(
15         sa.Integer, sa.ForeignKey(UserRole.id),
16         nullable=False, index=True)
17     ...
```

ОТНОШЕНИЯ

```
1 class UserRole(Base):
2     __tablename__ = 'user_role'
3
4     id = sa.Column(sa.Integer, primary_key=True)
5     name = sa.Column(sa.String, unique=True)
6     ...
7
8
9 class User(Base):
10     __tablename__ = 'user'
11
12     id = sa.Column(sa.Integer, primary_key=True)
13     email = sa.Column(sa.String, unique=True)
14     role_id = sa.Column(
15         sa.Integer, sa.ForeignKey(UserRole.id),
16         nullable=False, index=True)
17     ...
```

ОТНОШЕНИЯ

```
1 with create_session() as session:  
2     user = session.query(User).first()  
3     print(user.role_id)  
4 # 1
```

ОТНОШЕНИЯ

```
1 with create_session() as session:  
2     user = session.query(User).first()  
3     print(user.role_id)  
4 # 1
```

Чтобы получить сам объект **Role**, нужен ещё один запрос в базу?

ОТНОШЕНИЯ

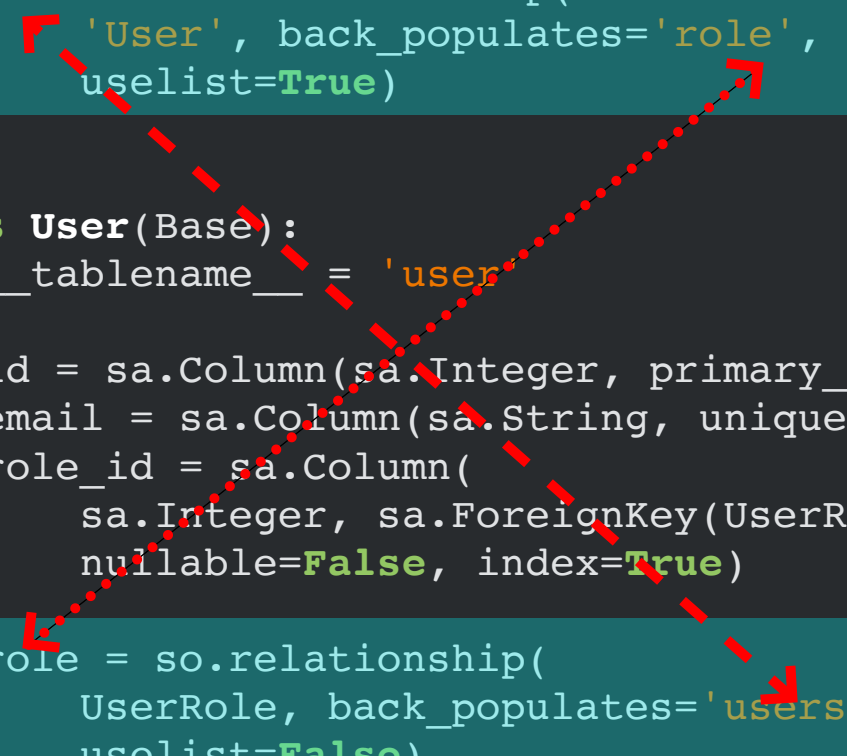
```
1 from sqlalchemy import orm as so
2
3 class UserRole(Base):
4     __tablename__ = 'user_role'
5
6     id = sa.Column(sa.Integer, primary_key=True)
7     name = sa.Column(sa.String, unique=True)
8
9     users = so.relationship(
10         'User', back_populates='role',
11         uselist=True)
12
13
14 class User(Base):
15     __tablename__ = 'user'
16
17     id = sa.Column(sa.Integer, primary_key=True)
18     email = sa.Column(sa.String, unique=True)
19     role_id = sa.Column(
20         sa.Integer, sa.ForeignKey(UserRole.id),
21         nullable=False, index=True)
22
23     role = so.relationship(
24         UserRole, back_populates='users',
25         uselist=False)
```

ОТНОШЕНИЯ

```
1 from sqlalchemy import orm as so
2
3 class UserRole(Base):
4     __tablename__ = 'user_role'
5
6     id = sa.Column(sa.Integer, primary_key=True)
7     name = sa.Column(sa.String, unique=True)
8
9     users = so.relationship(
10         'User', back_populates='role',
11         uselist=True)
12
13
14 class User(Base):
15     __tablename__ = 'user'
16
17     id = sa.Column(sa.Integer, primary_key=True)
18     email = sa.Column(sa.String, unique=True)
19     role_id = sa.Column(
20         sa.Integer, sa.ForeignKey(UserRole.id),
21         nullable=False, index=True)
22
23     role = so.relationship(
24         UserRole, back_populates='users',
25         uselist=False)
```

ОТНОШЕНИЯ

```
1 from sqlalchemy import orm as so
2
3 class UserRole(Base):
4     __tablename__ = 'user_role'
5
6     id = sa.Column(sa.Integer, primary_key=True)
7     name = sa.Column(sa.String, unique=True)
8
9     users = so.relationship(
10         'User', back_populates='role',
11         uselist=True)
12
13
14 class User(Base):
15     __tablename__ = 'user'
16
17     id = sa.Column(sa.Integer, primary_key=True)
18     email = sa.Column(sa.String, unique=True)
19     role_id = sa.Column(
20         sa.Integer, sa.ForeignKey(UserRole.id),
21         nullable=False, index=True)
22
23     role = so.relationship(
24         UserRole, back_populates='users',
25         uselist=False)
```



ОТНОШЕНИЯ

```
1 with create_session() as session:  
2     user = session.query(User).first()  
3     print(user.role.name)  
4 # admin
```


ОТНОШЕНИЯ

```
1 with create_session() as session:  
2     user = session.query(User).first()  
3     print(user.role.name)  
4 # admin
```

Миграции

Изменения схемы базы данных.

Для SQLAlchemy:
`pip install alembic`

alembic - инструмент генерации скриптов миграции
на основе описания классов.

Миграции

```
1 class User(Base):
2     __tablename__ = 'user'
3
4     id = sa.Column(sa.Integer, primary_key=True)
5     name = sa.Column(sa.String())
6
7     # НОВОЕ ПОЛЕ
8     email = sa.Column(sa.String())
```

Миграции

```
1 class User(Base):  
2     __tablename__ = 'user'  
3  
4     id = sa.Column(sa.Integer, primary_key=True)  
5     name = sa.Column(sa.String())  
6  
7     # НОВОЕ ПОЛЕ  
8     email = sa.Column(sa.String())
```

Миграции

```
1 class User(Base):
2     __tablename__ = 'user'
3
4     id = sa.Column(sa.Integer, primary_key=True)
5     name = sa.Column(sa.String())
6
7     # НОВОЕ ПОЛЕ
8     email = sa.Column(sa.String())
```

```
alembic revision -m "add user.email column"
```

Миграции

```
1  """add user.email column
2  Revision ID: 1975ea83b712
3  """
4
5  from alembic import op
6  import sqlalchemy as sa
7
8  # revision identifiers, used by Alembic.
9  revision = '1975ea83b712'
10 down_revision = None
11 branch_labels = None
12
13
14 def upgrade():
15     op.add_column('user', sa.Column('email', sa.String()))
16
17
18 def downgrade():
19     op.drop_column('user', 'email')
```

Миграции

накатываем 🍺 и откатываем

```
alembic upgrade head / <revision_id>  
alembic downgrade <revision_id>
```

I LOVE



TESTING!

Как тестировать

Как тестировать

- отдельная тестовая база;

Как тестировать

- отдельная тестовая база;
- создаём данные перед тестом;

Как тестировать

- отдельная тестовая база;
- создаём данные перед тестом;
- чистим базу между тестами.

Как тестировать

- отдельная тестовая база;
- создаём данные перед тестом;
- чистим базу между тестами.

```
1 from project.db import Base
2
3
4 @pytest.fixture(autouse=True)
5 def _init_db():
6     Base.metadata.create_all()
7     yield
8     Base.metadata.drop_all()
```

Вопросики