1 Создаем новый бд toDoList

```
CREATE DATABASE toDoList;
```

2 Создаем таблицу пользователей (user_data)

```
CREATE TABLE user_data (
    id BIGSERIAL NOT NULL PRIMARY KEY,
    email VARCHAR(40) NOT NULL UNIQUE,
    password VARCHAR(100) NOT NULL,
    name VARCHAR(50) NOT NULL
);
```

3 Создаем таблицу category

```
CREATE TABLE category (
    id BIGSERIAL NOT NULL PRIMARY KEY,
    title VARCHAR(50) NOT NULL,
    user_id BIGINT NOT NULL REFERENCES user_data(id)
);
```

4 Создаем таблицу priority

```
CREATE TABLE priority (
    id BIGSERIAL NOT NULL PRIMARY KEY,
    title VARCHAR(50) NOT NULL,
    color VARCHAR(30) NOT NULL,
    user_id BIGINT NOT NULL REFERENCES user_data(id)
);
```

5 Создаем таблицу задач (task)

```
CREATE TABLE task (
    id BIGSERIAL NOT NULL PRIMARY KEY,
    title TEXT NOT NULL,
    status BOOLEAN NOT NULL,
    date DATE,
    priority_id BIGINT REFERENCES priority(id),
    category_id BIGINT REFERENCES category(id),
    user_id BIGINT NOT NULL REFERENCES user_data(id)
);
```

6 Добавляем нового пользователя

```
INSERT INTO user_data (email,password,name)
VALUES ('yerezhepsultan11@gmail.com','password','Sultan');
```

7 Проверяем содержания таблицы user_data

```
SELECT * FROM user_data;
```

8 Добавляем новую категорию и проверяем содержимое

```
INSERT INTO category (title,user_id)
VALUES ('first category',1);
```

```
SELECT * FROM category;
```

	id [PK] bigint	title character varying (50)	user_id bigint
1	1	first category	1

9 Добавляем новый приоритет и проверяем содержимое

```
INSERT INTO priority (title,color,user_id)
VALUES ('first priority', 'red', 1);
```

```
SELECT * FROM priority;
```

	id [PK] bigint	title character varying (50)	color character varying (30)	user_id bigint
1	1	first priority	red	1

10 Добавляем новую задачу без приоритета, без даты и без категории

```
INSERT INTO task (title,status,user_id)
VALUES ('new task',false,1);
```

11 Добавляем новую задачу и проверяем содержимое

```
INSERT INTO task (title, status, date, priority_id, category_id, user_id)
VALUES ('new task', false,'2022-05-01', 1, 1, 1);
```

```
SELECT * FROM task
```

	id [PK] bigint	title text	status boolean	date date	priority_id bigint	category_id bigint	user_id bigint
1	1	new task	false	[null]	[null]	[null]	1
2	3	new task	false	2022-05-01	1	1	1

12 Обновляем статус первой задачи на true и проверяем содержимое

```
UPDATE task SET status = true WHERE id = 1;
```

id [PK] bigint	title text	status boolean	date date	priority_id bigint	category_id bigint	user_id bigint
3	new task	false	2022-05-01	1	1	1
1	new task	true	[null]	[null]	[null]	1

13 если попытаемся удалить пользователя

```
DELETE FROM user_data WHERE id = 1;
```

Получаем сообщение об ошибке

```
ERROR:  update or delete on table "user_data" violates foreign key constraint "priority_user_id_fkey" on table "priority"
DETAIL:  Key (id)=(1) is still referenced from table "priority".
SQL state: 23503
```

14 Чтобы пофиксить проблему нам нужно добавить (CASCADE ON DELETE для внешних ключей)

```
ALTER TABLE category DROP CONSTRAINT category_user_id_fkey
```

```
ALTER TABLE category ADD CONSTRAINT category_user_id_fkey
FOREIGN KEY (user_id)
REFERENCES user_data(id) ON DELETE CASCADE;
```

15 Удаляем пользователя

```
DELETE FROM user_data WHERE id = 1
```

```
DELETE 1

Query returned successfully in 56 msec.
```

```
SELECT * FROM user_data
```

id [PK] bigint	email character varying (40)	password character varying (100)	name character varying (50)

Когда мы удаляем пользователя удаляется все данные которые зависит от этого пользователя

```
SELECT * FROM category
```

id [PK] bigint	title character varying (50)	user_id bigint

```
SELECT * FROM priority
```

id [PK] bigint	title character varying (50)	color character varying (30)	user_id bigint

```
SELECT * FROM task
```

id [PK] bigint	title text	status boolean	date date	priority_id bigint	category_id bigint	user_id bigint

Вопросы

1. Что такое PK и в чем отличие от FK?

PK – Primary key не позволяет создавать одинаковых записей в таблица

FK – Foreign key – Внешний ключ - обеспечивает однозначную логическую связь между таблицами

2 Напишите все что знаете про объединение данных.

в языке SQL используется механизм объединения (join)

Есть две таблицы car и users

id [PK] bigint	make character varying (50)	model character varying (50)	price integer
2	BMW	3 Series	60013
3	Mercury	Cougar	74355
1	BMW	XK	51703

	id [PK] bigint	name character varying (40)	car_id bigint
1	1	Mark	1
2	2	John	3
3	4	Doe	[null]

1 INNER JOIN -внутреннее объединение таблиц

```
SELECT * FROM users
INNER JOIN car ON users.car_id = car.id
```

	id bigint	name character varying (40)	car_id bigint	id bigint	make character varying (50)	model character varying (50)	price integer
1	1	Mark	1	1	BMW	XK	51703
2	2	John	3	3	Mercury	Cougar	74355

2 LEFT JOIN или LEFT OUTER JOIN – левое внешнее объединение таблиц.

```
SELECT * FROM users
LEFT JOIN car ON users.car_id = car.id
```

	id bigint	name character varying (40)	car_id bigint	id bigint	make character varying (50)	model character varying (50)	price integer
1	1	Mark	1	1	BMW	XK	51703
2	2	John	3	3	Mercury	Cougar	74355
3	4	Doe	[null]	[null]	[null]	[null]	[null]

3 RIGHT JOIN или RIGHT OUTER JOIN – правое внешнее объединение таблиц.

```
SELECT * FROM users
RIGHT JOIN car ON users.car_id = car.id
```

	id	name character varying (40)	car_id bigint	id bigint	make character varying (50)	model character varying (50)	price integer
1	1	Mark	1	1	BMW	XK	51703
2	2	John	3	3	Mercury	Cougar	74355
3	[null]	[null]	[null]	2	BMW	3 Series	60013

3. Какие существуют связи таблиц?

Один к одному (one to one)

```sql
CREATE TABLE users (
    id BIGSERIAL NOT NULL PRIMARY KEY,
    name VARCHAR(40) NOT NULL,
    car_id BIGINT REFERENCES car(id) UNIQUE
)
```

Один к многим (one to many)

```sql
CREATE TABLE users (
    id BIGSERIAL NOT NULL PRIMARY KEY,
    name VARCHAR(40) NOT NULL,
    car_id BIGINT REFERENCES car(id)
)
```

Много к многим (many to many)

```sql
CREATE TABLE user_products (
    id BIGSERIAL NOT NULL PRIMARY KEY,
    user_id BIGINT NOT NULL REFERENCES users(id),
    product_id BIGINT NOT NULL REFERENCES products(id)
)
```

4 Какие бывают агрегатные функции ?

Avg(column) – среднее значение

```sql
SELECT AVG(price) as avg_price FROM car
```

	avg_price numeric
1	9333.3333333333333333

Max(column) – максимальное значение

```
SELECT MAX(price) FROM car
```

	max numeric
1	17000.00

Mix(column) – минимальное значение

```
SELECT MIN(price) FROM car
```

	min numeric
1	4000.00

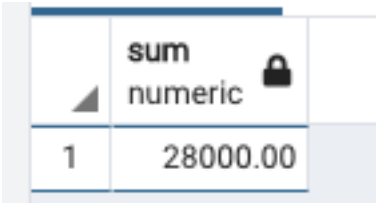
COUNT (*) или COUNT(column) – количество строк

```
SELECT COUNT(*) FROM car
```

	count bigint
1	3

SUM(column) – сумма переданного столбца

```
SELECT SUM(price) FROM car
```



	sum numeric
1	28000.00

5 Напишите все что знаете про группировку с примерами запросов.

GROUP BY - Группировка по столбцу

```
SELECT make FROM car GROUP BY make;
```

Ford
Maserati
Dodge
Infiniti
Holden
Audi
Lexus
Jeep
Cadillac
Ferrari

Удобно использовать вместе с агрегатными функциями

```
SELECT make,COUNT(*) FROM car GROUP BY make;
```

make character varying (50)	count bigint
Ford	90
Maserati	5
Dodge	48
Infiniti	12
Holden	1
Audi	32
Lexus	18
Jeep	11
Cadillac	20

Можно сделать group by сразу по нескольким column

```
SELECT make,model,COUNT(*) FROM car GROUP BY make,model;
```

	make character varying (50)	model character varying (50)	count bigint
1	Toyota	Land Cruiser	6
2	Mercedes-Benz	S-Class	1
3	Infiniti	FX	1
4	Saturn	Aura	1
5	Kia	Amanti	1
6	Daewoo	Lanos	1
7	Subaru	Forester	4
8	Ford	Econoline E350	3
9	Ferrari	612 Scaglietti	1

6 Что такое 1НФ, 2НФ, 3НФ (1NF, 2NF, 3NF)?

1 НФ

1. Определяет необходимые данные, которые становятся колонками таблицы. Размещает связанные по смыслу данные в таблице.
2. Гарантирует, что в БД нет повторяющихся групп данных.
3. Гарантирует, что каждая таблица содержит первичный ключ (primary Key).

2 НФ
В базе данных колонки таблиц не должно быть частичной зависимости от первичного ключа
3 НФ
Все не главные поля зависят от первичного ключа

8 Можно ли объединить 3 (4, 5, <2) таблицы?

```
SELECT user_data.id, user_data.name, priority.title, category.title FROM user_data
INNER JOIN category ON user_data.id = category.user_id
INNER JOIN priority ON user_data.id = priority.user_id
```

9 Сделать запрос где есть:

- выборка: функция агрегации + обычные поля

```
SELECT make, COUNT(*) FROM car GROUP BY make
```

make character varying (50)	count bigint
GMC	54
Maybach	1
Lincoln	15
Honda	28
Daewoo	3

- группировка по нескольким полям

```
SELECT make,model FROM car GROUP BY make,model
```

make character varying (50)	model character varying (50)
Toyota	Land Cruiser
Mercedes-Benz	S-Class
Infiniti	FX
Saturn	Aura
Kia	Amanti
Daewoo	Lanos

- join нескольких таблиц

```
SELECT user_data.id, user_data.name, priority.title, category.title FROM user_data
INNER JOIN category ON user_data.id = category.user_id
INNER JOIN priority ON user_data.id = priority.user_id
```

- условие where

```
SELECT * FROM car WHERE price > 60000
```

id [PK] bigint	make character varying (50)	model character varying (50)	price integer
2	BMW	3 Series	60013
3	Mercury	Cougar	74355
4	Chrysler	Town & Country	73407
5	Chevrolet	Uplander	90301
6	Volvo	C30	71726
7	Mercedes-Benz	CLK-Class	94068
8	Ford	Taurus	92095

- сортировка результата

```
SELECT * FROM car WHERE price > 60000 ORDER BY model DESC
```

id [PK] bigint	make character varying (50)	model character varying (50)	price integer
454	Scion	tC	60381
477	Scion	tC	68249
726	Audi	riolet	79840
752	Volkswagen	rio	64733
774	Isuzu	i-280	91362
729	Honda	del Sol	99210
622	BMW	Z4	91575

10 **Индексы** создаются для тех полей, которые участвуют в условиях каких? (запросы)

Индексы используется для ускорения получения данных

```
SELECT * FROM car WHERE id = 1
```

id [PK] bigint	make character varying (50)	model character varying (50)	price integer
1	Jaguar	XK	51703

11Чтобы вручную обновить индексы для нужной таблицы, нужно выполнить команду _____

```
          ^
[jasacademy=# CREATE INDEX ON cities(title);
CREATE INDEX
```

```
        ^
jasacademy=# \d cities
                                    Table "public.cities"
 Column |          Type          | Collation | Nullable |              D
--------+------------------------+-----------+----------+---------------
-------------
 id     | bigint                 |           | not null | nextval('cities
'::regclass)
 title  | character varying(100) |           | not null |
Indexes:
    "cities_pkey" PRIMARY KEY, btree (id)
    "cities_title_idx" btree (title)
```

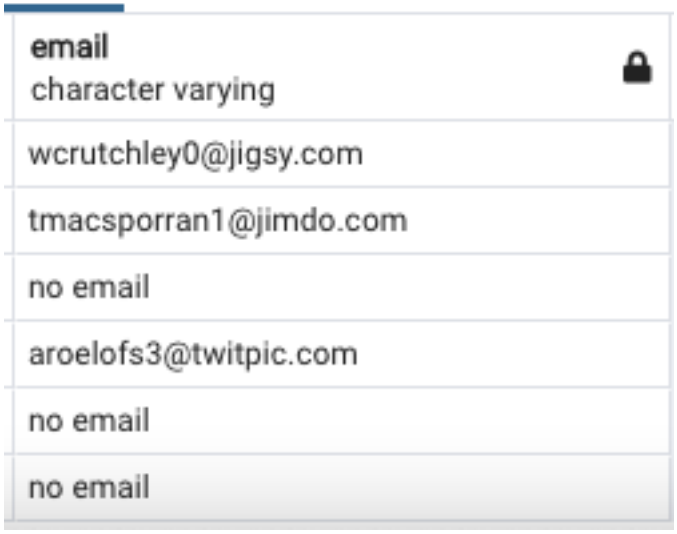
12 Какая может быть причина(/ы) создания индексов в вашей БД?

Когда в бд много данных.
Дает возможность быстро находить результат поиска

13Что делает оператор **coalesce?** Приведите пример запроса.

Функция COALESCE возвращает первый попавшийся аргумент, отличный от NULL. Если же все аргументы равны NULL, результатом тоже будет NULL. Это часто используется при отображении данных для подстановки некоторого значения по умолчанию вместо значений NULL

```sql
SELECT COALESCE(email, 'no email') as email FROM person
```



email character varying
wcrutchley0@jigsy.com
tmacsporran1@jimdo.com
no email
aroelofs3@twitpic.com
no email
no email

14Существует ли преобразование с одного типа в другой в Postgresql?

CAST (выражение AS тип)

15 Напишите одну функцию и один триггер. Код и скрины БД вставить.

```
CREATE OR REPLACE FUNCTION log_car_make_changes()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
  AS
$$
BEGIN
    IF NEW.make <> OLD.make THEN
         INSERT INTO car_make_changes_log(old_make,new_make)
         VALUES(OLD.make,NEW.make);
    END IF;

    RETURN NEW;
END;
$$
```

```
CREATE TRIGGER make_changes
  BEFORE UPDATE
  ON car
  FOR EACH ROW
  EXECUTE PROCEDURE log_car_make_changes();
```

```
UPDATE car SET make = 'BMW' WHERE id = 1
```

```
SELECT * FROM car_make_changes_log
```

id [PK] bigint	old_make character varying (40)	new_make character varying (40)
1	Jaguar	BMW

16 Напишите транзакцию и объясните в чем отличие от обычного запроса.

В транзакциях мы должны подтвердить запрос с помощью COMMIT

```
BEGIN;
UPDATE accounts SET balance = balance - 1500 WHERE id = 1;

UPDATE accounts SET balance = balance + 1500 WHERE id = 2;

COMMIT;
```

Или можно отменить изменение с помощью ROLLBACK;

```sql
BEGIN;
UPDATE accounts SET balance = balance - 1500 WHERE id = 1;

UPDATE accounts SET balance = balance + 1500 WHERE id = 2;

ROLLBACK;
```

17 Что такое ACID?

Транзакция PostgreSQL является атомарной, согласованной, изолированной и надежной. Эти свойства часто называют ACID (atomic, consistent, isolated, and durable):

Атомарность гарантирует, что транзакция завершается по принципу «все или

Согласованность гарантирует, что изменение данных, записываемых в базу данных, должно быть действительным и соответствовать предопределенным правилам.

Изоляция определяет, насколько целостность транзакции видна другим транзакциям.

Долговечность гарантирует, что транзакции, которые были зафиксированы, будут постоянно храниться в базе данных.
ничего».