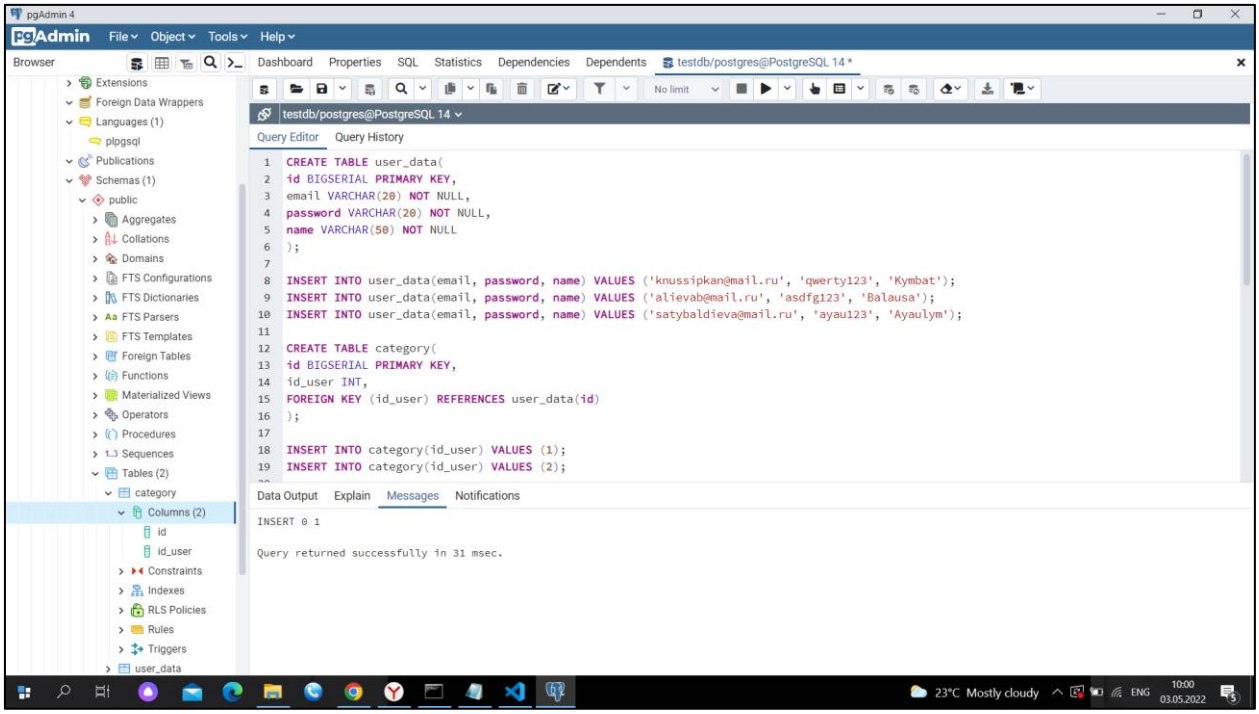
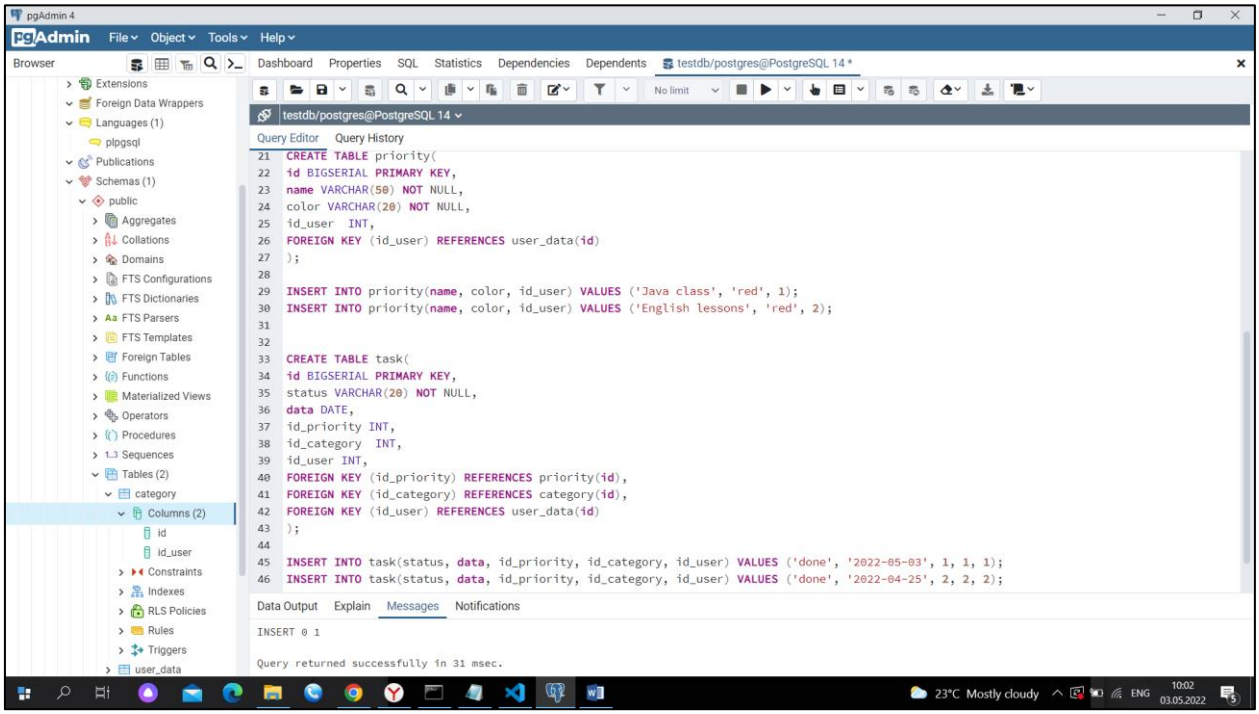
Нусипкан Кымбат

CREATE TABLE





UPDATE TABLE

```
49  UPDATE priority SET color = 'green' WHERE id = 2;
50  select * from priority where id = 2;
51
52
53
```

Data Output | Explain | Messages | Notifications

	id [PK] bigint	name character varying (50)	color character varying (20)	id_user integer
1	2	English lessons	green	2

```
54  UPDATE task SET data = '2022-05-05', status = 'not done' WHERE id = 1;
55  select * from task t, user_data u
56  where t.id_user = u.id AND t.id = 1;
57
```

Data Output | Explain | Messages | Notifications

	id bigint	status character varying (20)	data date	id_priority integer	id_category integer	id_user integer	id bigint	email character varying (20)	password character varying (20)	name character varying (50)
1	1	not done	2022-05-05	1	1	1	1	knussipkan@mail.ru	qwerty123	Kymbat

```
13  UPDATE user_data SET email = 'nusipkank@gmail.com' WHERE id=1;
14  SELECT * FROM user_data WHERE id=1;
15
```

Data Output | Explain | Messages | Notifications

	id [PK] bigint	email character varying (20)	password character varying (20)	name character varying (50)
1	1	nusipkank@gmail.com	qwerty123	Kymbat

DELETE TABLE

Невозможно удалить данные с таблицы, если они связаны между собой. В нашем случае таблица задач привязана к таблице категории. Для того чтобы удалить столбец, сначала его нужно удалить с таблицы задач.

```
28  DELETE FROM category WHERE id = 2;
29
```

Data Output | Explain | Messages | Notifications

```
ERROR: ОШИБКА:  UPDATE или DELETE в таблице "category" нарушает ограничение внешнего ключа "task_id_category_fkey" таблицы "task"
DETAIL:  На ключ (id)=(2) всё ещё есть ссылки в таблице "task".


SQL state: 23503
```

```
30  DELETE FROM task WHERE id = 2;
31
```

Data Output | Explain | Messages | Notifications

```
DELETE 1

Query returned successfully in 29 msec.
```

```
28  DELETE FROM category WHERE id = 2;
29
```

Data Output | Explain | Messages | Notifications

```
DELETE 1

Query returned successfully in 26 msec.
```

INNER JOIN

```
SELECT tt.id, tt.status, tt.data, tt.id_priority, pt.name, pt.color, tt.id_category, ct.id_user,
       u.email, u.name
FROM task tt
INNER JOIN user_data u ON tt.id_user = u.id
INNER JOIN category ct ON tt.id_category = ct.id
INNER JOIN priority pt ON tt.id_priority = pt.id
WHERE tt.status = 'not done';
```

Data Output | Explain | Messages | Notifications

	id bigint	status character varying (20)	data date	id_priority integer	name character varying (50)	color character varying (20)	id_category integer	id_user integer	email character varying (20)	name character varying (50)
1	1	not done	2022-05-05	1	Java class	red	1	1	nusipkank@gmail.com	Kymbat

```
SELECT * FROM priority
INNER JOIN user_data ON priority.id_user = user_data.id
WHERE color = 'red';
```

Data Output | Explain | Messages | Notifications

	id bigint	name character varying (50)	color character varying (20)	id_user integer	id bigint	email character varying (20)	password character varying (20)	name character varying (50)
1	3	English lessons	red	1	1	nusipkank@gmail.com	qwerty123	Kymbat
2	1	Java class	red	1	1	nusipkank@gmail.com	qwerty123	Kymbat

LEFT JOIN

```
SELECT tt.id, tt.status, tt.data, tt.id_priority, pt.name, pt.color, tt.id_category, ct.id_user,
       u.email, u.name
FROM task tt
LEFT JOIN user_data u ON tt.id_user = u.id
LEFT JOIN category ct ON tt.id_category = ct.id
LEFT JOIN priority pt ON tt.id_priority = pt.id;
```

Data Output | Explain | Messages | Notifications

	id bigint	status character varying (20)	data date	id_priority integer	name character varying (50)	color character varying (20)	id_category integer	id_user integer	email character varying (20)	name character varying (50)
1	1	not done	2022-05-05	1	Java class	red	1	1	nusipkank@gmail.com	Kymbat
2	3	done	[null]	3	English lessons	red	1	1	nusipkank@gmail.com	Kymbat
3	7	done	2022-04-25	2	English lessons	green	3	2	alievab@mail.ru	Balausa

ОТВЕТЫ НА ВОПРОСЫ

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

PK – это primary key. Т.е. внешний ключ, с помощью которого мы будем связывать между собой таблицы.

FK – это foreign key, нужен для привязки таблицы, то есть primary key.

Например, у нас есть две таблицы: Company, Employee. Соответственно в этих таблицах есть два PK, это – id_employee и id_company. Мы понимаем, что определенный человек работает в определенной компании. И мы привязываем работника к компании с помощью FK.

2. Напишите все что знаете про объединение данных.

В приведенном выше примере у нас есть две таблицы. И мы хотим получить данные о компании, и о сотрудниках, которые там работают. Для этого мы объединяем данные с помощью JOIN. Для получения конкретной информации, т.е. например, о сотрудниках, кто работает в компании Jusan мы используем INNER JOIN. Также у нас могут быть сотрудники, которые

уволились и еще не устроились на работу, и мы также хотим получить и их данные. В этом случае мы используем LEFT JOIN.

3. Какие существуют связи таблиц? дайте примеры (создайте таблицы со связями) и скрины прикрепите к ответам

Есть три вида связи: one to one, one to many, many to many. Объясню на примерах:

- one to one – один сотрудник может работать только в одной компании

```
create table company(
id BIGSERIAL PRIMARY KEY,
name varchar(50) not null
);

create table employee(
id BIGSERIAL PRIMARY KEY,
name varchar(50) not null,
id_company INT REFERENCES company(id),
CONSTRAINT EMPL_COMP_ID UNIQUE (id_company)
);


INSERT INTO company(name) VALUES ('Jusan');
INSERT INTO employee(name, id_company) VALUES ('John', 3);

SELECT * FROM employee
INNER JOIN company ON employee.id_company = company.id;
```

Data Output | Explain | Messages | Notifications

	id bigint	name character varying (50)	id_company integer	id bigint	name character varying (50)
1	3	John	3	3	Jusan

- one to many – один сотрудник может работать в нескольких компаниях

```
create table company(
id BIGSERIAL PRIMARY KEY,
name varchar(50) not null
);

create table employee(
id BIGSERIAL PRIMARY KEY,
name varchar(50) not null,
id_company INT REFERENCES company(id)
);


INSERT INTO company(name) VALUES ('Jusan');
INSERT INTO employee(name, id_company) VALUES ('John', 3);

SELECT * FROM employee
INNER JOIN company ON employee.id_company = company.id;
```

Data Output | Explain | Messages | Notifications

	id bigint	name character varying (50)	id_company integer	id bigint	name character varying (50)
1	3	John	3	3	Jusan

- many to many – один сотрудник может иметь несколько навыков, в свою очередь один навык может принадлежать нескольким сотрудникам

```
116 create table skill(
117 id BIGSERIAL PRIMARY KEY,
118 name varchar(50) not null
119 );
120
121 create table employeeSkills(
122 id_e INT,
123 id_s INT,
124 FOREIGN KEY(id_e) REFERENCES employee(id),
125 FOREIGN KEY(id_s) REFERENCES skill(id)
126 );
127
128 INSERT INTO employee(name, id_company) VALUES ('John', 3);
129 INSERT INTO employee(name, id_company) VALUES ('Kate', 3);
130 INSERT INTO employee(name, id_company) VALUES ('Dana', 3);
131 INSERT INTO skill (name) VALUES ('Java');
132 INSERT INTO skill (name) VALUES ('SQL');
133 INSERT INTO employeeSkills (3, 2);
134 INSERT INTO employeeSkills (4, 1);
135 INSERT INTO employeeSkills (5, 1);
136 INSERT INTO employeeSkills (5, 2);
```

4. Какие бывают агрегатные функции?

- MIN() – находит минимальное значение

```
postgres=# SELECT MIN(price) FROM Car;
  min
-------
 10089
(1 row)
```

- MAX() – находит максимальное значение

```
postgres=# SELECT MAX(price) FROM Car;
   max
-------
 99956
(1 row)
```

- AVG() – находит среднюю арифметическую

```
postgres=# SELECT AVG(price) FROM Car;
        avg
--------------------
 54954.250000000000
(1 row)
```

- SUM() – находит сумму всех чисел

```
postgres=# SELECT SUM(price) FROM Car;
   sum
----------
 54954250
(1 row)
```

- COUNT() – возвращает количество строк

```
postgres=# SELECT COUNT(price) FROM Car WHERE make = 'Toyota';
 count
-------
    49
(1 row)
```

5. Напишите все что знаете про группировку с примерами запросов.

GROUP BY – группирует данные таблицы по условии, которые мы указываем. Например, нам нужно найти количество машин принадлежаший к определенной марке, т.е. сколько моделей из определленой марки у нас есть в базе. В этом случае запрос будет таким:

```
postgres=# SELECT COUNT(*), make FROM CAR
postgres-# GROUP BY(make);
 count |     make
-------+---------------
    96 | Ford
     7 | Maserati
    39 | Dodge
    16 | Infiniti
     4 | Bentley
     1 | Austin
     1 | Rambler
    32 | Pontiac
     4 | Plymouth
     1 | Holden
    19 | Audi
    24 | Lexus
    12 | Jeep
    25 | Volvo
    27 | Cadillac
    14 | Acura
    21 | Suzuki
     6 | Aston Martin
     1 | Citro┼лn
    33 | Mitsubishi
     1 | Foose
     3 | Ferrari
    13 | Chrysler
     9 | Isuzu
```

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

- 1NF – это когда, одна ячейка содержит лишь одно значение и каждая запись должна быть уникальной. Например таблица с данными актеров и данные о фильмах, в котором снимался данный актер (все в одной таблице).

- 2NF. Для преобразования с 1НФ в 2НФ – разделяем таблицу. То есть оставляем в первой таблице лишь данные об актере, и первичным ключом сделаем id_actor. И создаем еще одну таблицу с фильмами и с id_actor. Тут мы используем one to many, где one – это актер, а many – это фильмы. То есть один актер может сниматься в нескольких фильмах.
- 3NF. Для преобразования в 3НФ мы уже должны быть на 2НФ. И самое главное не должны иметь транзитивных функциональных зависимостей транзитивных. Для лучшего понимания, давайте создадим новую таблицу с данными менеджеров. И в таблицу актеров добавим FK менеджеров. То есть у каждого актера есть свой менеджер (посчитаем, что один менеджер может работать с несколькими актерами).

7. В чем отличие materialized view от обычного view?

Тут я бы сказала, что в хранении. То есть view не сохраняется нигде, ни на диске, ни на таблице, соответственно не занимает место в хранилище. Если в таблице происходит update, каждый раз мы будем получать обновленные данные, когда будем запрашивать view.

В свою очередь materialized view хранится в диске. Если в таблице происходит update, обновленные данные мы не получим с его помощью.

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

Да, изи.

```
71  SELECT tt.id, tt.status, tt.data, tt.id_priority, pt.name, pt.color, tt.id_category, ct.id_user,
72             u.email, u.name
73  FROM task tt
74  INNER JOIN user_data u ON tt.id_user = u.id
75  INNER JOIN category ct ON tt.id_category = ct.id
76  INNER JOIN priority pt ON tt.id_priority = pt.id
77  WHERE tt.status = 'not done';
```

Data Output | Explain | Messages | Notifications

	id bigint	status character varying (20)	data date	id_priority integer	name character varying (50)	color character varying (20)	id_category integer	id_user integer	email character varying (20)	name character varying (50)
1	1	not done	2022-05-05	1	Java class	red	1	1	nusipkank@gmail.com	Kymbat

9. Сделать запрос где есть:

- выборка: функция агрегации + обычные поля
- группировка по нескольким полям
- join нескольких таблиц
- условие where
- сортировка результата

```sql
SELECT COUNT(*), pt.color, tt.status
FROM task tt
INNER JOIN user_data u ON tt.id_user = u.id
INNER JOIN category ct ON tt.id_category = ct.id
INNER JOIN priority pt ON tt.id_priority = pt.id
WHERE data > '2022-04-01'
GROUP BY(pt.color, tt.status)
ORDER BY pt.color;
```

Data Output | Explain | Messages | Notifications

	count bigint	color character varying (20)	status character varying (20)
1	1	green	done
2	1	red	not done

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

Индексы создаются для тех записей, которых юзер часто ищет. А для первичного и уникального ключа индекс создается автоматом. Соответственно они занимают место в хранилище. Еслы мы часто ищем машин именно по марке, мы можем для марки создать индекс для повышения производительности.

```
postgres=# CREATE INDEX ON car(make);
CREATE INDEX
postgres=# \d car
                    Table "public.car"
 Column |         Type          | Collation | Nullable | Default
--------+-----------------------+-----------+----------+---------
 id     | integer               |           |          |
 make   | character varying(50) |           |          |
 model  | character varying(50) |           |          |
 price  | integer               |           |          |
Indexes:
    "car_make_idx" btree (make)
    "unique_car_id" UNIQUE CONSTRAINT, btree (id)
Referenced by:
    TABLE "salesman" CONSTRAINT "salesman_car_id_fkey" FOREIGN KEY (car_id) REFERENCES car(id)
```

11. Чтобы вручную обновить индексы для нужной таблицы,нужно выполнить команду ALTER INDEX index_name REBUILD;
12. Какая может быть причина(/ы) создания индексов в вашей БД?

Во-первых, для повышения производительности. Когда мы будем работать с огромной базой данных, введение результатов запроса будет занимает некоторые время, пока SQL не проидется во всем строкам. Вот в таких случаях индексы ускоряют процесс. Сейчас пользу индексов мы можем и не ощущать, из-за базы которую мы сейчас имеем (мало данных).

13. Что делает оператор coalesce? Приведите пример запроса.

Я так понимаю, этот оператор заменяет значение полей. Например, у нас в таблице есть null значения на поле email. И мы хотим убрать null-ы, и вместо этого вставить запись ‘no email’.

```
postgres=# SELECT COALESCE(model, 'nothing') FROM car;
       coalesce
----------------------
 G
 1500
 Mazda6
 Ram 2500
```

14. Существует ли преобразование с одного типа в другой в Postgresql? Если да, то как это делать?

Да. Оно нужно в случае слияния двух таблиц. Например, один и тот же столбец может быть находится разные типы данных. В таких случаях, мы воспользуемся преобразованием. Вот некоторые из них:

- bit.toint(b) — приведение битовой последовательности в десятичное представление
- to_char(t.g) — преобразование типа «timestamp» в строку формата «g»
- to_date(s.g) — преобразование строки типа даты «g» в значение типа «date»;
- to_number(s.g) — преобразование строки типа даты «g» в значение типа «numeric»;
- to_timestamp(s.g) — преобразование строки формата даты «g» в значение типа «timestamp»;
- timestamp(d) — преобразование типа «data» в тип «timestamp».

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

```
12 CREATE OR REPLACE FUNCTION test()
13   RETURNS trigger AS
14 $$
15 BEGIN
16          INSERT INTO user_data(email, password, name)
17          VALUES(NEW.email, NEW.password, NEW.name);
18
19     RETURN NEW;
20 END;
21 $$
22 LANGUAGE 'plpgsql';
23
24 CREATE TRIGGER test_trigger
25   AFTER INSERT
26   ON user_data
27   FOR EACH ROW
28   EXECUTE PROCEDURE test();
29
30 INSERT INTO user_data(email, password, name) VALUES ('admin@mail.ru', 'qwerty123', 'Alima');
31
```

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

Лучше всего использовать транзакцию для банковского дела. Там есть переводы денег. Взяв это в основу, мы можем понять, что во время перевода денег с одного счета на другой происходит транзакция. А с помощью обычного запроса такое не сделаешь. Также можно откатить транзакцию, как приведенном ниже примере. Сперва я внесла 2 изменении, а затем отменила их с помощью rollback. А если захочу сохранить изменения, то используем commit transaction. А запросы это запросы, выводит результаты лишь того, что ты написал.

```
268  BEGIN;
269  UPDATE user_data
270  SET email = 'knussipkan'
271  WHERE id = 1;
272
273  UPDATE user_data
274  SET password = 'admin123'
275  WHERE id = 3;
276
277  ROLLBACK;
278
```

Data Output | Explain | Messages | Notifications

```
ROLLBACK

Query returned successfully in 26 msec.
```

```
121  select * from user_data;
122
```

Data Output | Explain | Messages | Notifications

	id [PK] bigint	email character varying (20)	password character varying (20)	name character varying (50)
1	2	alievab@mail.ru	asdfg123	Balausa
2	3	satybaldieva@mail.ru	ayau123	Ayaulym
3	1	nusipkank@gmail.com	qwerty123	Kymbat

Editable column

17. Что такое ACID?

ACID – это гарантия БД, чтобы поддерживать транзакции. Т.е. БД гарантирует изолированность транзакции. Давайте расшифруем термин, и попытаемся полностью его понять.

Атомарность – гарантия того, что никакая транзакция в системе не будет выполнена частично, и что все его подоперации будут выполнены. Поскольку невозможно одновременно выполнить все операции, вводится термин «откат транзакции». То есть, если транзакцию не удастся полностью выполнить, отменяются все изменения.

Согласованность – это транзакция, достигающая своего нормального решения. Например, для банковских переводов, сумма для перевода не должна превышать сумму на счете и т.д.

Изолированность – это гарантия того, что во время выполнения транзакции параллельные транзакции не должны оказывать влияния на её результат.

Прочность – это гарантия того, что изменения сохраненные после успешной транзакции, не должны подвергаться опасности (не сохраняются) не смотря на проблемы нижнего уровня.