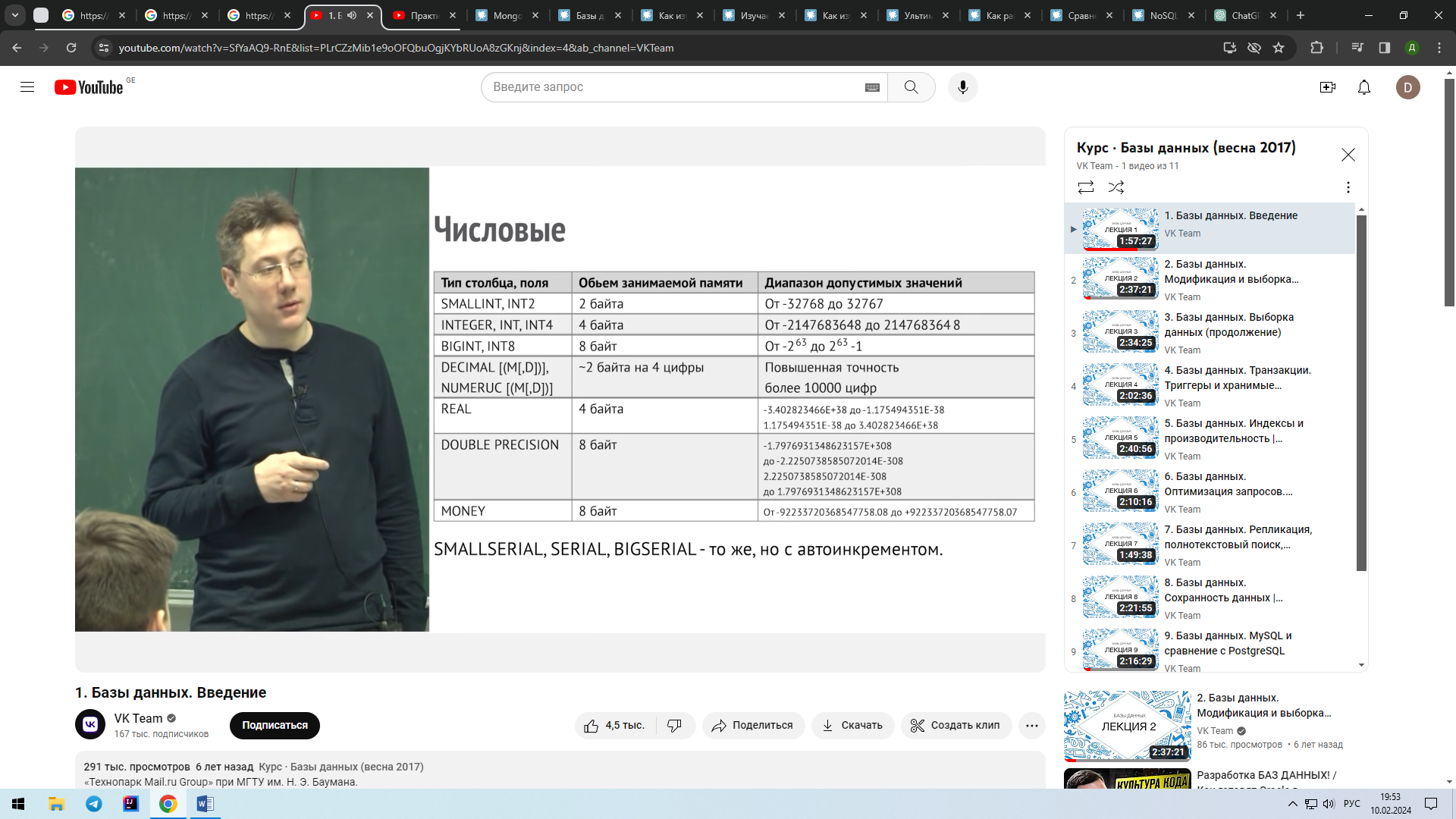
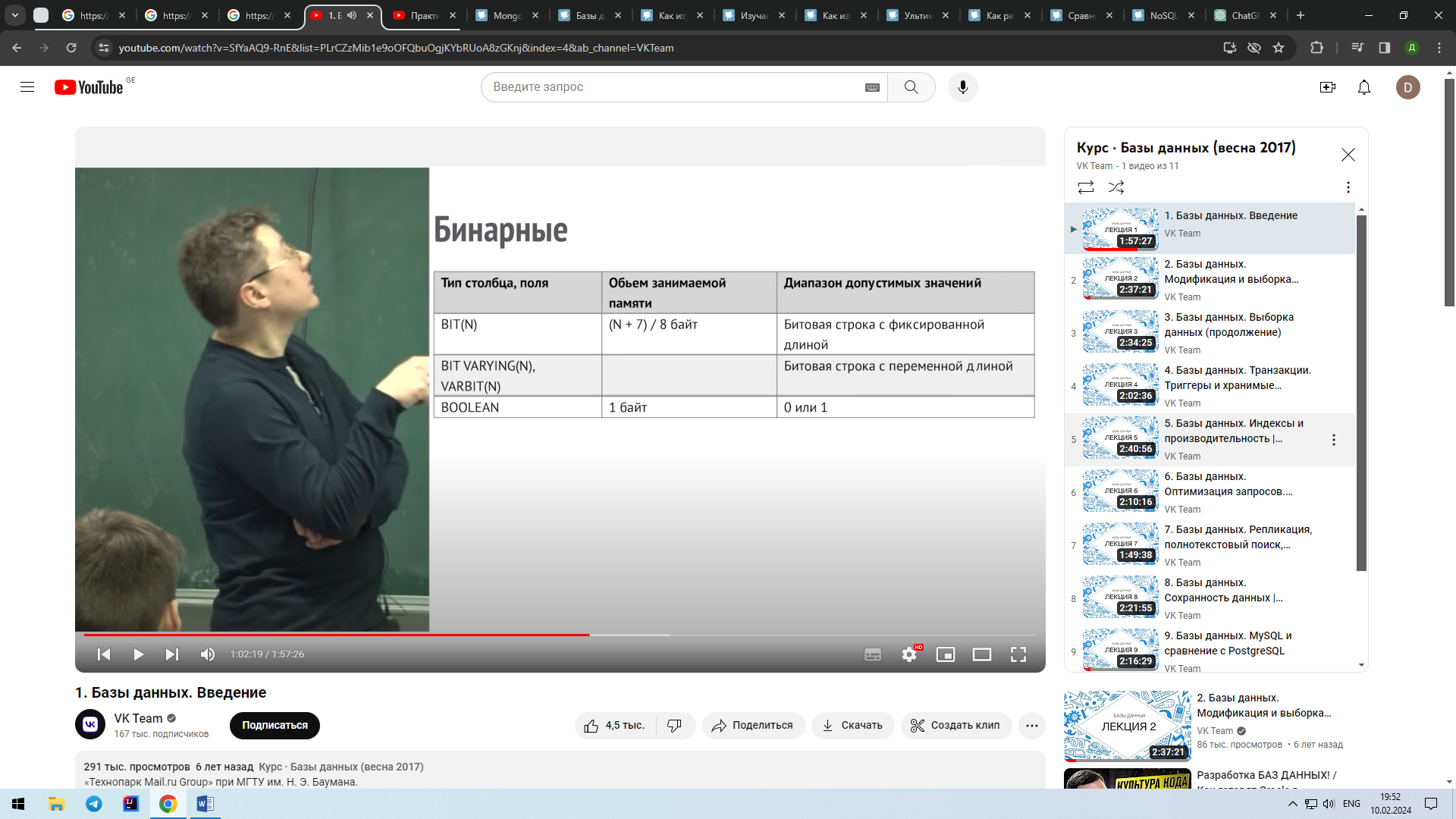
**Типы данных в PostgreSQL**

*Числовые*

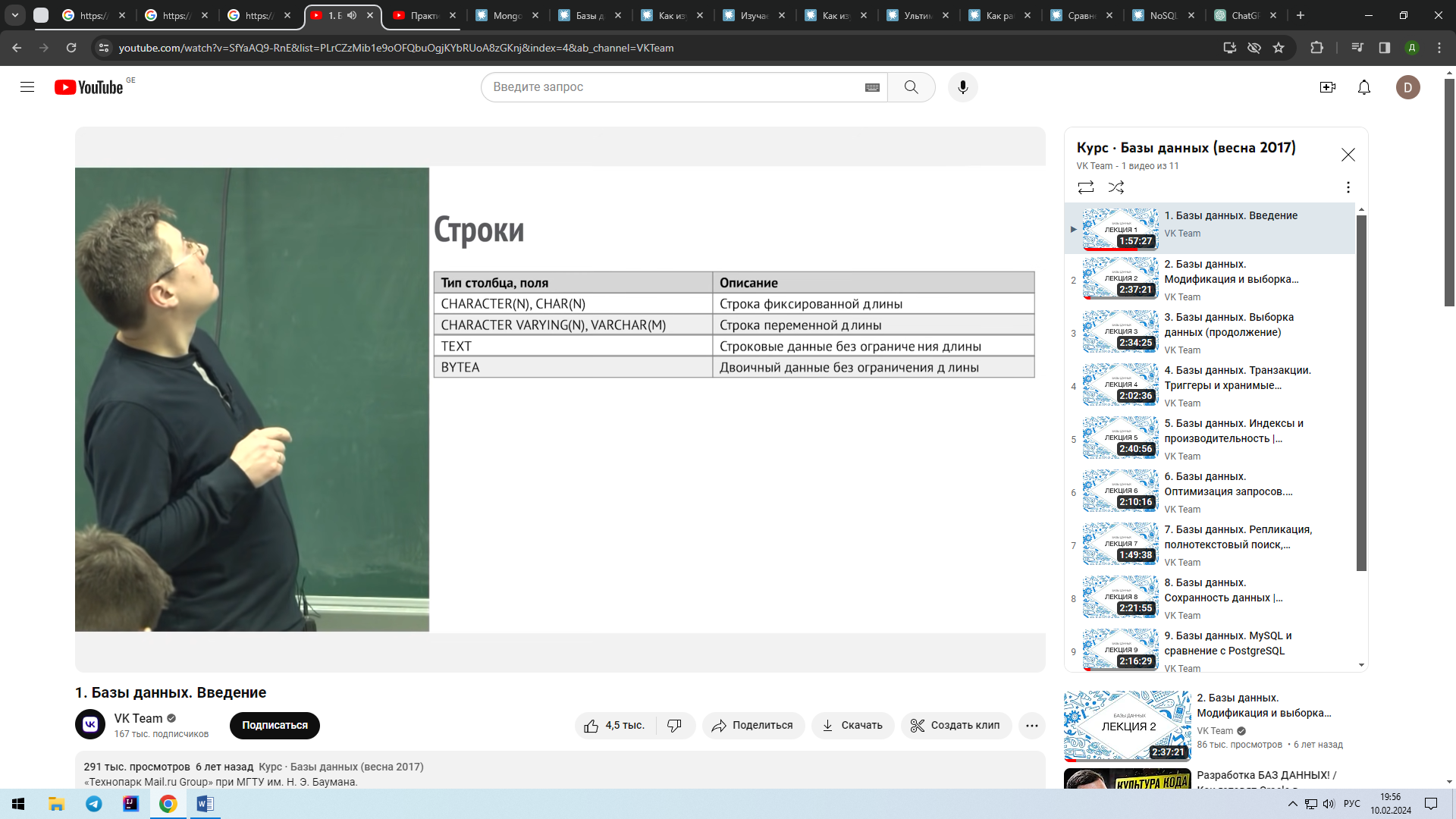
*decimal/numeric* –используется для обозначения денежных значений (можно использовать тип MONEY, но он менее точен) m –максимальное общее кол-во цифр, d – максимальное кол-во цифр после запятой. То есть целая часть может содержать не больше m-d цифр. Если d больше допустимого, то оно будет округлено до допустимого.

Serial – беззнаковые, автоинкрементируемые (под капотом Sequence)



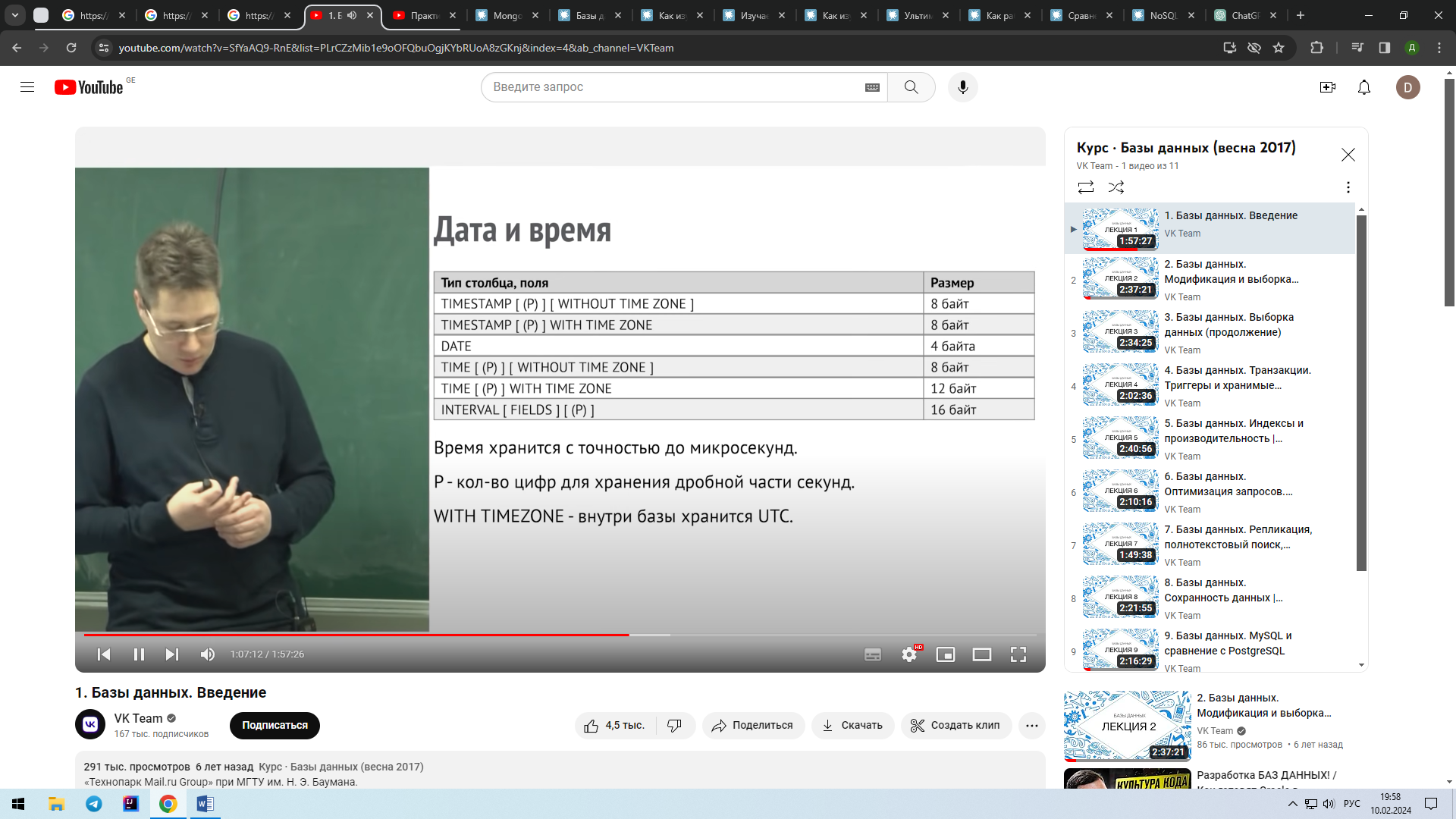
Bit(n) – фиксированная длина, если меньше – дописываются нули.

Varbit(n) – фиксированная длина, если меньше – нули не дописываются



Сhar(n) – фикс длина, если меньше – дописываются пробелы

Varchar(n) – фикс длина, если меньше – пробелы не дописываются – более оптимальный вариант в большинстве случаев



Date хранит число, месяц год и время (часы, минуты, секунды)

Timestamp – Date + миллисекунды

Null – значение без типа или отсутствие значения

Поддерживает Arrays, JSON, XML, геометрические типы, custom-типы.

**Категории запросов:**

[DDL](#DDL) (Data Definition Language) – CREATE, ALTER, DROP, TRUNCATE – команды для определения и изменения структуры БД.

[DML](#DML) (Data Manipulation Language) – SELECT, INSERT, UPDATE, DELETE, MERGE – команды для манипулирования данными внутри таблицы.

DCL (Data Control Language) – GRANT, REVOKE, DENY - команды для управления правами доступа и безопасностью данных.

[TCL](#TSL) (Transaction Control Language) - COMMIT, ROLLBACK, SEVEPOINT - Команды для управления транзакциями.

Distinct – возвращает уникальные значения атрибута.

**Select distinct** column1 **from** table\_name; - выведет уникальные значения для column1

**Select distinct** column1, column2 **from** table\_name; - Выведет уникальные комбинации значений для column1 и column2

SELECT column1 DISTINCT column2 FROM table\_name; - Такой синтаксис недопустим по причине возможного несоответствия кол-ва значений двух атрибутов (у первого выведутся все значения, включая неуникальные, у второго неуникальные отбрасываются, что ведет к несоответствию кол-ва строк и неопределенностью в заполнении строк данными)

As – алиас, используется для задания псевдонима столбца или таблицы в результирующем запросе (относится к получаемому выводу, сама таблица не меняется), оператор можно опускать.

**Select \* from** table **as** new\_name - для переименования таблицы (можно использовать в запросе для сокращения синтаксиса)

**Select** column **as** new\_name **from** table - для переименования колонок в результирующем выводе.

Join– Оператор для объединения данных из нескольких таблиц путем сопоставления значений в указанных столбцах и «наложения» кортежей этих таблиц друг на друга.

**Select** **\* from** table1

**join** table2 **on** table1.column = table2.column

Using – синтаксический сахар, используется вместо ON, в случае, если имена атрибутов, по которым соединяются таблицы, совпадают.

**Select** **\* from** table1

**join** table2 **using** (column)

Limit – Ограничивает выборку результатов

Offset – используется вместе с LIMIT, задает отправную точку для оператора LIMIT

**Select** \* **from** employees

**limit** 10 **offset** 20 – запрос вернет 10 строк, начиная с 21й строки

Fetch – команда для работы с курсором, используется для извлечения строк из набора данных, на которые указывает курсор.

Order by - задает сортировку вывода.

**Order by** <Аргумент > **<asc / desc>** **<nulls first / last>**

ASC (ascending order) – сортировка по возрастанию (по умолчанию)  
 DSEC (descending order) – сортировка по убыванию

Аргумент сортировки может быть задан Атрибутом, алиасом или номером атрибута. Несколько аргументов для сортировки определяют более точный порядок сортировки (сортировка по первому аргументу, для повторяющихся значений первого аргумента сортировка по второму аргументу).

Where – фильтрация строк по условию, выбираются те строки, которые соответствуют условию.

Group by - группирует строки на основании одного или нескольких атрибутов (создаст подгруппы с уникальными комбинациями этих значений). Т.е. собирает строки с одинаковым значением указанных атрибутов в одну группу, для каждой такой группы могут быть применены только агрегатные функции.

В Select могут использоваться только атрибуты, включенные в group by или результат агрегатных функций этой группы.

**Select** department, name, **SUM** (salary)

**From** employees

**Group** **by** department;

Атрибут name не может быть включен в выборку, т.к. он не участвует в группировке и его множество может превышать множество department.

Grouping sets – расширяет функционал group by, позволяет явно указывать сразу несколько атрибутов или их комбинаций для группировки в рамках одного запроса (Также добавляет кортеж, отражающий результат всех наборов группировок). Требует явно указать подгруппы для группировок.

**Select** column\_1, column\_2, agg\_function  
 **from** table\_name  
 **group** **by** **grouping** **sets** ((column\_1), (column\_1, column\_2));

Группировка произойдет по column\_1 и для каждой группы будет вычислен результат agg\_function и по column\_1, column\_2 и также будет вычислен результат агрегирующей функции для этой группы (в конце результат агрегирующей функции для всей табилцы).

Rollup – также создает комбинации группировок, сначала группировка по первому аргументу, затем первый + второй, затем первый + второй + третий и т.д. (Также добавляет кортеж, отражающий результат всех наборов группировок).

**Select** column\_1, column\_2, agg\_function  
 **from** table\_name  
 **group** **by rollup** (column\_1, column\_2);

Группировка произойдет сначала по первому аргументу (column\_1), затем по первому + второй (column\_1, column\_2).

Cube – создает все варианты комбинаций подгрупп из указанных аргументов (Также добавляет кортеж, отражающий результат всех наборов группировок).

**Select** column\_1, column\_2, agg\_function  
 **from** table\_name  
 **group** **by cube** (column\_1, column\_2);

Группировка произойдет по column\_1, затем по column\_2, затем по column\_1 + column\_2.

Having – оператор для фильтрации групп, которые используются в group by.

Where – применяются к исходным данным, которые находятся в таблице (сначала выполняется он).

Having – эти условия запускаются после того как сформированы группы

*Expressions* – результаты вывода для каждого аргумента можно изменять перед выводом посредством операций и выражений (изменяется вывод, но база не меняется).

Арифметические операции, операции сравнения, текстовые операции (конкатенация), логические (and, or, not), функции (sum, avg, count и тд) и др.

1) Арифметические операции - Подразумевают операции сложения, вычитания, умножения, деления, процентный остаток. Могут быть произведены как между аргументом и числовыми значениями, так и между аргументами.

2) Текстовые операции – Основная текстовая операция – конкатенация, с помощью оператора ||, помимо нее есть функции для работы со строками типа concat, upper, lower, length, substring, trim, left, right. (см. функции)

3) Операции сравнения – стандартные (>, <, = и тд), null операции (is null, is not null) так же можно отнести к операциям сравнения, но так как любое сравнение с null дает false (даже с другим null) их вынесли отдельно. При сравнении текстовых значений используется лексикографический порядок (как в словаре).

4) Логические операции используются для комбинирования нескольких операций сравнения – and, or, not.

and – вводит несколько условий для соответствия.   
or – вводит условия, одно из которых должно быть выполнено  
not – исключает случаи, подходящие под условия (where not <условие>).

5) Специфические операторы SQL:

Оператор between - задает промежуток значений. Является аналогом записи

**Where** <аргумент> **between** значение\_1 **and** значение\_2

Оператор in (список значений) – проверка на совпадение значения с некоторым списком

Является сокращенным аналогом оператора « = »

**where** <Атрибут > = <значение> **or** <Атрибут > = <значение>

Оператор Like – оператор для поиска среди текстовых типов по заданному шаблону. Для поиска используются маски:

% - любое кол-во любых символов

\_ - один любой символ

[A-B] – диапазон допустимых символов

[^A-B] или [!A-B] – отрицание этого диапазона

| - или в регулярных выражениях

Оператор Ilike – регистронезависимый поиск.

Оператор Escape – вводит экранирующий символ.

**LIKE** ‘%25’ **ESCAPE** ‘2’ – экранирует символ % (он будет рассматриваться не как специальный символ).

6) Функции

|  |  |  |
| --- | --- | --- |
| ***single-row*** - кол-во аргументов соответствует кол-ву результатов | | |
| Lower/Upper() | Преобразуют строку в нижний/верхний регистр | |
| left/right() | Left – возвращает первые n символов, right – последние n символов | |
| Initcap() | Преобразует первую букву каждого слова в верхний регистр | |
| Concat() | Объединяет строки (аналог ||) | |
| Length() | Возвращает длину строки | |
| Lpad(a, b, c)  Rpad(a, b, c) | используется для дополнения строки слева/справа символами до определенной длины (a – модифицируемая строка, b – длина результирующей строки, c – добавляемые символы) | |
| Trim() | Удаляет заданный символ с начала - leading, конца – trailing или двух концов – both слова.  Trim (a b from c) – a – модификатор удаления, b – символ, который нужно удалить, c – откуда удалить. Если не задать модификатор, по умолчанию будет both, если не задать символ, то по умолчанию будет пробел. | |
| Position() | Возвращает индекс первого совпадения с подстрокой для символа.  Position(‘substring’ in ‘string’) | |
| Substring() | Возвращает подстроку из заданного диапазона.  Substr(‘string’, index, length) – string - источник поиска, index – индекс начала поиска (если аргумент отрицательный - будет вестись отчет с конца), length – кол-во символов (необязательный аргумент)  Так же можно использовать синтаксис Substr(‘string’ from index for length) | |
| Replace() | Заменяет искомые символы на заданные.  Replace(a, b, c) – a – строка для произведения замены, b – подстрока, которую заменяем, c – подстрока, на которую заменяем (для удаления вставляется пустая строка) | |
| Round()  Ceil() вверх до целого  Floor() - вниз | Функция для округления значений.  Round(a, b) – a – число для округления, b – точность (кол-во цифр после запятой) | |
| Trunk() | Отсекает значения, без округления.  Trunk(a, b) – число для «округления», b – точность (если отрицательное, то отсечется целая часть, но сохранятся разряды, отсутствие аргумента отсекает дробную часть)  Пример:   select trunc (123.45, 1) вывод 123.4  Select trunc(123.45, -2); вывод: 100 | |
| Mod() | Функция для нахождения остатка от деления Mod(делимое, делитель) | |
| Функции для работы с датой  \*Математические операции с датой дают число дней\*  Математические операции между датами запрещены (кроме минуса)  Дробные числа так же преобразуются в интервал дней + часы | | |
| Now()  Current\_timestamp | Возвращает время и дату сервера. | |
| Extract | Извлекает компоненты из даты и времени  Extract( field from source) – field – year, month, day, hour и тд. | |
| Interval | Для прибавления или вычитания определенного интервала к нужной дате  Interval задается с помощью ‘<число><единица измерения времени>’  Пример: Select <Дата> + Interval ‘1 year’; | |
| Next\_Day | Вычисляет ближайший заданный день недели по отношению к заданной дате (будущий).  В Postgre нет такой функции, можно исполнить запросом, но он не учитывает кейс, когда дата текущая дата имеет искомый день недели:  SELECT ‘some\_date'::date + ( 4 + 7 - extract ( dow FROM ‘some\_date'::date))::int%7; – приведение типов DATE нужно использовать, если используешь строки, а не текущую дату – 4 порядковый номер четверга. | |
| Last\_day | Показывает последний день месяца (сколько дней в месяце)  В Potgre нет такой функции, запрос:  SELECT EXTRACT(DAY FROM (DATE\_TRUNC('MONTH', CURRENT\_DATE)  +INTERVAL '1 MONTH - 1 day')) | |
| Date\_trunc | Date\_trunc(Интервал, источник) – усекает дату до указанного интервала. Интервал указывается в формате ‘day’, ‘week’ и тд. | |
| Coalesce() | Возвращает первое ненулевое значение из множества аргументов (минимум 2). | |
| Nullif() | Сравнивает 2 аргумента и возвращает null, если они равны, если нет, то возвращает первый аргумент. | |
| Условные функции | | |
| Case | Simple case | Searched case |
| Одно выражение сравнивается с несколькими условиями. | Каждый кейс может быть представлен разными типами выражений. |
| Case <выражение>  When value1 then result1  When value2 then result2  ………  Else result3  Условий должно быть минимум одно, значения value должны быть совместимы со значением выражения | Case  When <выражение> условие then resilt1  When <выражение> условие then resilt2  …….  Else result3  Каждое выражение может быть уникальным, как и условие. |
| Case записывается как колонка в выборке: select first, sec, case when….. | |
| *multiple-row* (агрегатные функции) – функции, обрабатывающие набор значений и возвращающие итоговое значение. | | |
| Count() | Возвращает кол-во строк (может быть использован с Distinct)  count(\*) – посчитает все строки, включая null  count(column) – посчитает строки not null  count(‘static arg’) – создаст новую колонку и заполнит ее значениями static arg и вернет кол-во строк.  \*select ‘static agr’ from some\_table создает новый атрибут и заполняет каждую ячейку значением статического аргумента. | |
| Sum() | Sum(distinct/all { столбец/выражение })  Игнорирует null (любая арифметическая операция с null равняется null)  При написании выражения типа select sum(100+100) from students; - сначала вычислится аргумент 200, затем добавится новый столбец и заполнится значениями 200, затем каждый из них просуммируется и результат будет зависеть от кол-ва строк в таблице students | |
| Avg() | Вычисление среднего арифметического (так же имеет модификатор distinct/all) | |
| Max()/Min() | Поиск максимального и минимального значений. Работает как с числами, так и с датами и строковыми типами (лексикографический порядок).  Max/Min(distinct/all {столбец/выражение}) – distinct/all не имеет смысла, т.к. не влияет на результат | |
| Nested group функции | Не имеет смысла использовать вложенные агрегирующие функции, так как результат этих функций не будет отличаться от первоначального.  Глубина single-row функций не ограничена. | |

**DML** ([back](#Категории))

**Select –** команда для извлечения информации из БД.

**Insert into –** Добавление новых строк в таблицу.

Insert into <Название таблицы> (<Названия атрибутов>) values (<Значения атрибутов>)

Возможно добавление нескольких строк одной командой.

**Insert** **into** employees (id, name, gross) **values** (1, ‘Ivan’, 1000), (2, ‘Petr’, 2000)

(<Названия атрибутов>) – использование необязательно, явно задает порядок вставки значений, если не использовать, то будет использован порядок как в таблице.

Возможно использование конструкции **ON** **CONFLICT** <action>, которая задает поведение при конфликтных вставках, задается для атрибутов с ограничениями. **Action** конфигурируется опциями **Do** **Nothing** – не выполнять никаких действий, включая выброс исключений, **Do** **Update Set –** обновление строки при возникновении конфликта (необходимо сконфигурировать обновление строки путем указания полей для обновления).

**Insert** **into** employees (id, name, gross)   
**values** (1, ‘Ivan’, 1000), (2, ‘Petr’, 2000)  
**on** **conflict** (id) **Do** **update** **set** name = **Excluded**.name, gross = **Excluded**.gross

**Excluded –** псевдоним для строки, которая вызвала конфликт.

Добавленные значения должны соответствовать типам колонок и Constraint наложенные на эти атрибуты.

Совместно с оператором может использоваться Select, для добавления значений из других таблиц, добавляться значения по умолчанию или добавляться сразу несколько строк в одном запросе.

**Update** – обновление существующих записей в таблице (включает 2 команды Insert и Delete).

По умолчанию изменяет все строки таблицы по указанному атрибуту или атрибутам, для частичного изменения таблицы вводятся условия с помощью where.

**Update** <Название таблицы> **set** <Атрибут> = <значение>

**where** <условие>;

**Delete** – осуществляет построчное удаление всех совпадающих с условием строк, если условия не заданы, то удаляются все строки.

**Delete from** <Название таблицы> **where** <условие>

**Returning** – возвращает данные, которые были вставлены, изменены или удалены (работает как select, путем перечисления необходимых к возврату атрибутов).

**Truncate** – DDL команда для быстрого удаления всех данных таблицы путем изменения ее метаданных (перемещается указатель high water mark), а не построчного прохождения по каждой удаляемой строке.

**Truncate table** employees;

[Разница между truncate и delete](https://info-comp.ru/differences-between-truncate-and-delete-in-t-sql)

**TSL** ([back](#Категории))

Незафиксированные изменения видимы только в текущей сессии.

**Begin –** обозначение начала транзакции.

**Commit –** Команда для фиксации изменений в рамках текущей транзакции. **Rollback** – Отмена изменений, сделанных в рамках текущей транзакции.

**Savepoint**<имя точки восстановления> – установка точки восстановления в текущей транзакции.

**Rollback to sevepoint** <точка восстановления>– Отмена изменений до указанной точки восстановления.

**DDL** ([back](#Категории))

|  |  |
| --- | --- |
| CREATE | TABLE |
| INDEX <name> on <table\_name>(column) |
| VIEW |
| SEQUENCE |
| DATABASE |
| SCHEMA |
| FUNCTION |
| PROCEDURE |

|  |  |  |
| --- | --- | --- |
| ALTER | DATABASE |  |
| INDEX |  |
| VIEW |  |
| TABLE | ADD/DROP COLUMN |
| ALTER COLUMN SET DATA TYPE |
| RENAME old\_column TO new\_column |
| RENAME TO |
| ADD/DROP CONSTRAINT |
| ADD/PROP INDEX |
| OWNER TO |
| SET SCHEMA |
|  | USER |  |
|  | SCHEMA |  |
|  | FUNCTION |  |
|  | TRIGGER |  |

|  |  |
| --- | --- |
| DROP | TABLE |
| INDEX |
| VIEW |
| DATABASE |
| COLUMN |

COMMENT: Добавление комментариев к объектам базы данных.

*\*Приведены примерный синтаксис и возможности, подробнее – смотри доку\**