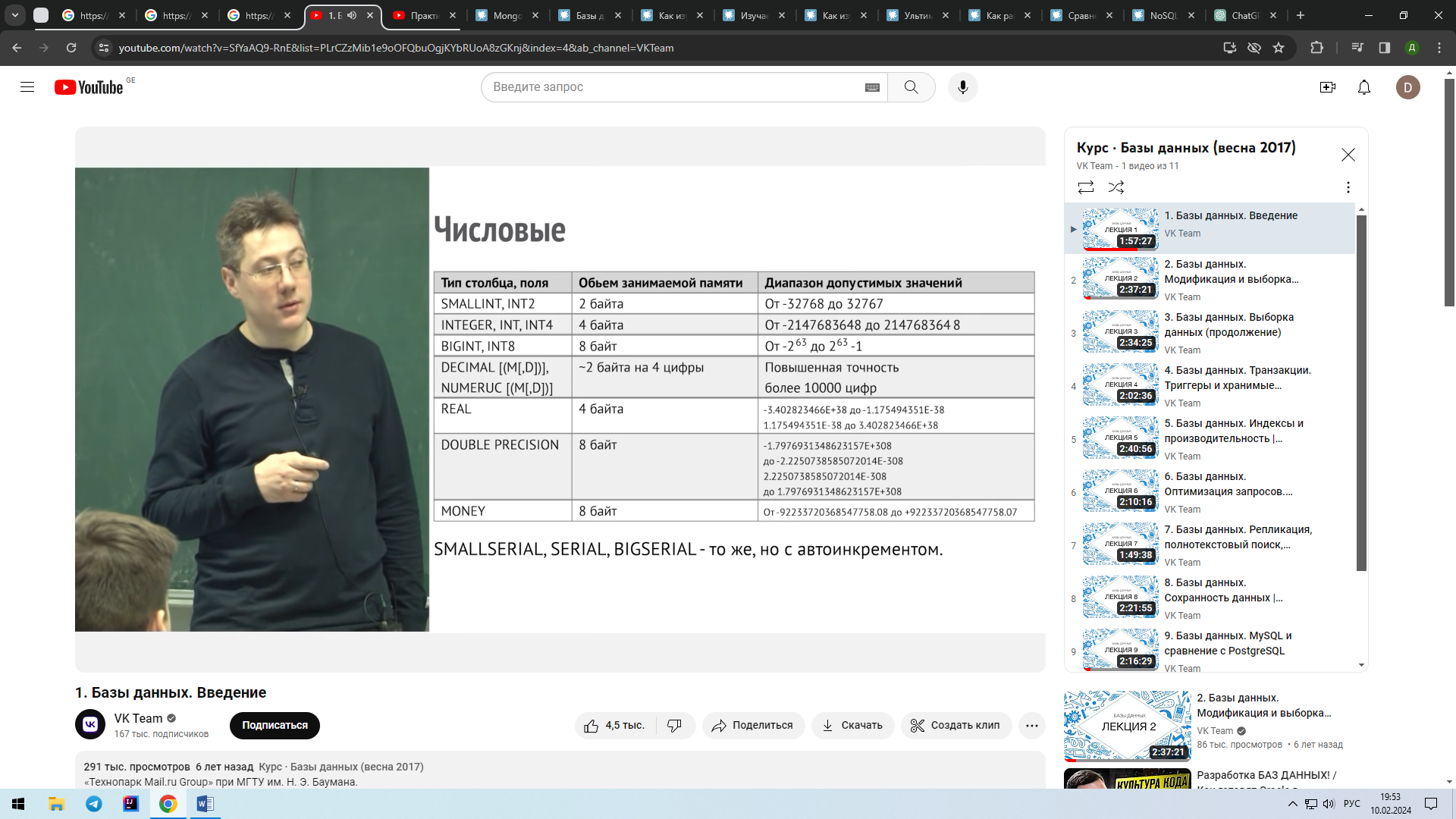
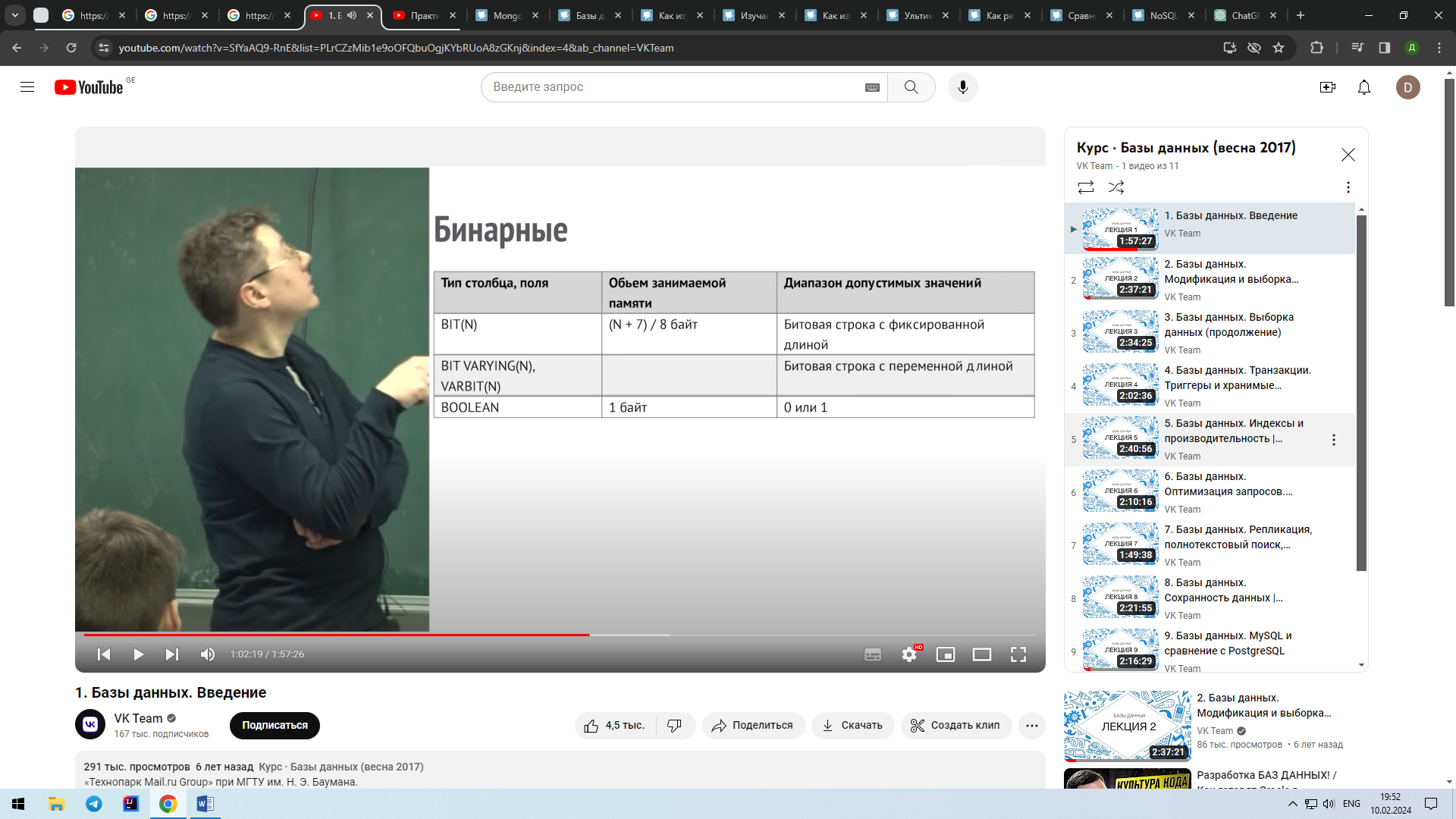
**Типы данных в PostgreSQL**

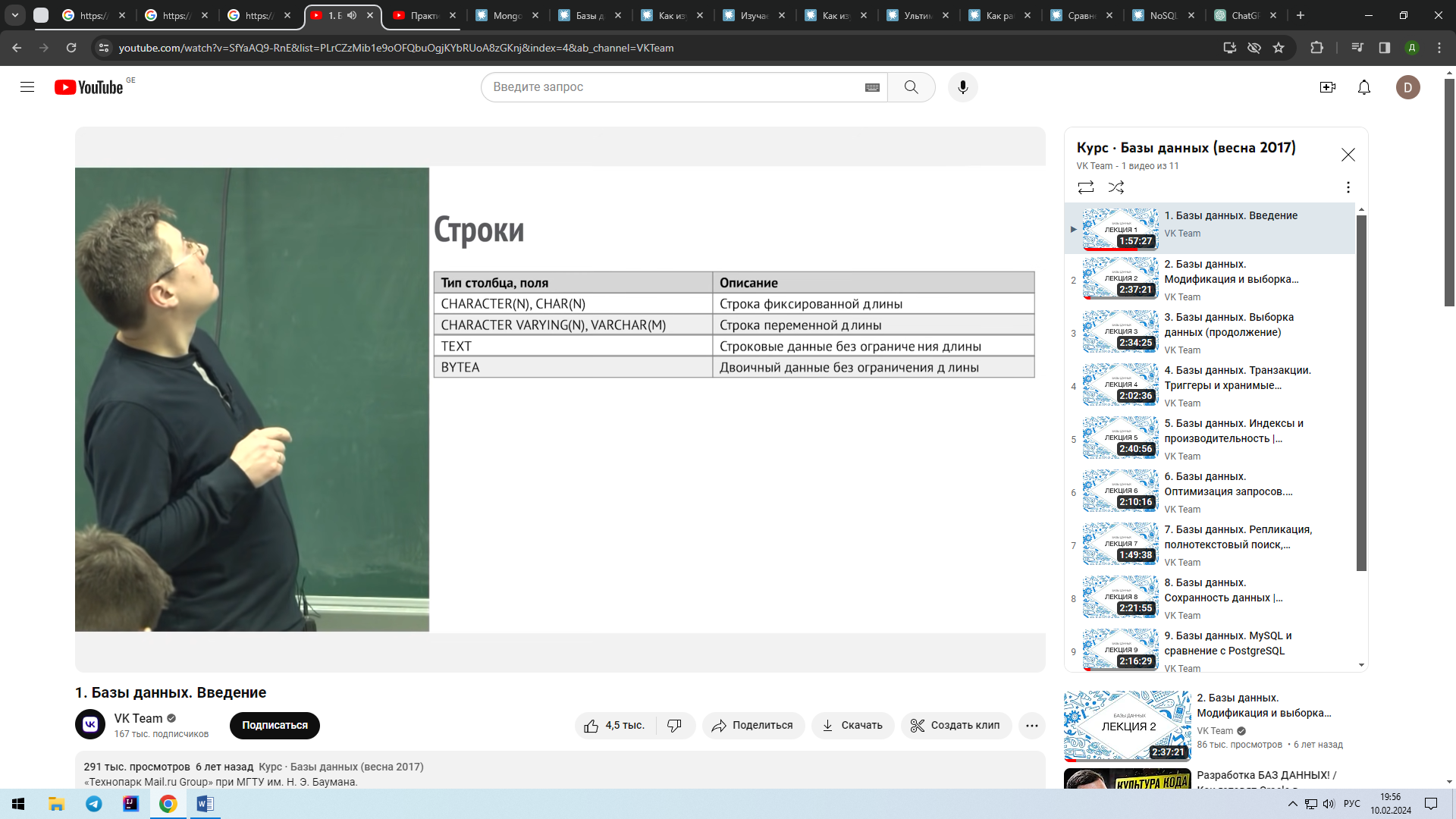
*Числовые*

*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

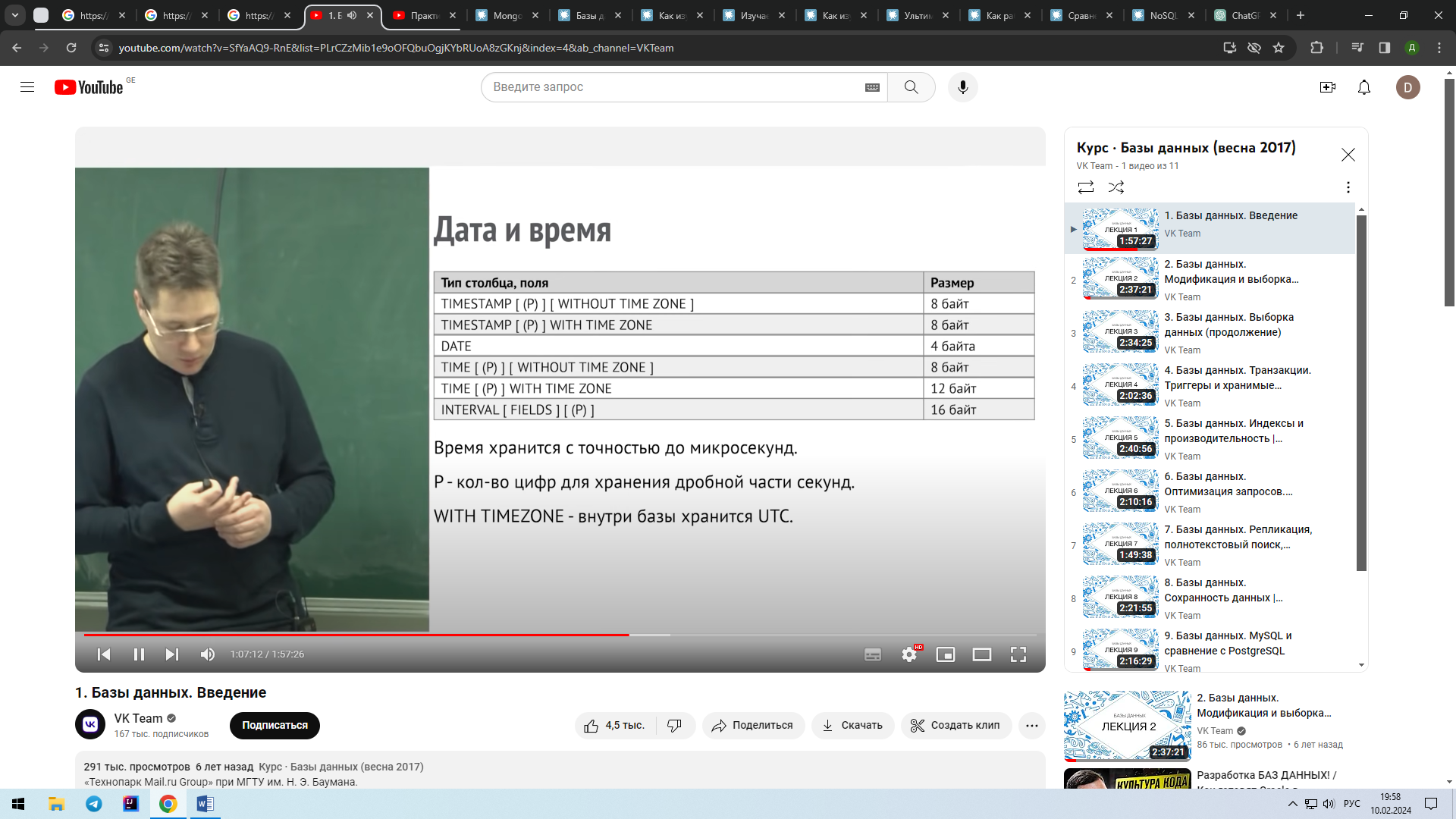


Bit(n) – фиксированная длина, если меньше – дописываются нули.

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.

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

**Distinct** – возвращает уникальные кортежи.

**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.

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

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

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

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

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

*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.

Операции сравнения, логические операции и null операции: Эти типы выражений используются в контексте оператора Where, с помощью которого задаются условия результирующей выборки значений.

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>).

Оператор between так же задает промежуток значений. Является аналогом записи

Where аргумент between table\_1 >= значение and <= значение;

Оператор in(список значений) является аналогом оператора =, выводит значения, соответствующие одному из значений в списке, может быть заменен на where <колонка> = <значение> or <колонка> = <значение>

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

| - или

Escape – вводит символ для экранирования, например LIKE 'ST$\_%' ESCAPE '$'; - здесь \_ означает не оператор для like, а символ, который нужно найти.

Ilike – регистронезависимый поиск.

Limit – Ограничивает выборку результатов

5) Функции

|  |  |  |
| --- | --- | --- |
| *single-row* - кол-во аргументов соответствует кол-ву результатов | | |
| Lower() |  | |
| Upper() |  | |
| Initcap() | первая буква каждого слова будет иметь верхний регистр, остальные нижний (разделителями выступают пробелы и спец символы). | |
| Concat() | аналог ||, | |
| Length() |  | |
| 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’ [from start\_position]) - substring – искомая строка, string – источник поиска, [from start\_position] – начиная с какого символа искать | |
| Substring() | Возвращает подстроку из заданного диапазона.  Substr(a, b, c) – a – источник поиска, b – индекс начала поиска (если аргумент отрицательный - будет вестись отчет с конца), c – кол-во символов (необязательный аргумент)  Так же можно использовать синтаксис Substr(a from b for c) | |
| Replace() | Заменяет искомые символы на заданные.  Replace(a, b, c) – a –источник для поиска, b – что заменяем c – на что заменяем.  Для удаления нужных символов c задается как ‘’; | |
| 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, не входящие в диапазон точности (без округления)  Trunk(a, b) – число для «округления», b – точность (если отрицательное, то отсечется целая часть, но сохранятся разряды)  Пример: 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(<Нужная размерность> from <Источник>) | |
| Interval | Для прибавления или вычитания определенного интервала к нужной дате  Interval ‘<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) – интервал указывается в формате ‘day/week etc’ | |
| 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) - результат всегда один, независимо от кол-ва аргументов. Групповые функции не могут быть написаны вместе с другими аргументами: ~~select count(name), id from students;~~ (потому что не совпадает кол-во строк). | | |
| Count() | Возвращает кол-во строк, значение которых отлично от null  Count(distinct/all { \*/столбец/выражение }) – all добавляется по умолчанию  Select ‘asd’ from students; выведет asd для каждой колонки, если передать такой статичный аргумент в count(‘asd’) from students; то он сначала создаст отдельный столбец для asd и заполнит каждую строку этим значением, а затем посчитает | |
| 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) | |
| Group by() | Группирует строки по уникальным значениям. Как бы создает подтаблицы для каждой группы значений, что позволяет использовать другие multiple-функции  Сочетание group by и других multiple-функции всегда вычисляет результат для каждой подтаблицы уникального значения.  Написание group by с несколькими параметрами создаст подгруппы с уникальными комбинациями этих значений | |
| 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 | |
| Having | Отсеивает сгруппированные результаты по условию. Работает как where.  Where – применяется к отдельным строкам перед группировкой (применяется для фильтрации строк), т.е. в качестве условия должны выступать неагрегированные значения.  Having – применяется к подтаблицам (группам) после группировки (применяется для фильтрации групп), в качестве условия могу выступать только multiple-функции. | |
| 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. | |

Order by <аргумент> **<asc / desc>** **<nulls first / last>–** задает сортировку вывода. Asc, desc

Аргумент может быть именем аргумента, выражением, алиасом, номером аргумента в select

Несколько аргументов для сортировки – определяет более точный порядок сортировки. Если по первому аргументу значения повторяются, то они по сути никак не будут отсортированы и для таких случаев можно использовать другой аргумент, по которому они так же будут отсортированы

**DML** (кроме select). ([back](#Категории))

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

Insert into <Название таблицы > (<Названия колонок>) values(<Значения столбцов>)

(<Названия колонок>) – использование необязательно, вручную обозначает, в какие колонки будут добавляться данные и в каком порядке, если не использовать, то будет использован порядок как в таблице.

Добавленные значения должны соответствовать типам колонок и ограничениям (Constraint) на значения, наложенные на эти колонки.

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** – для обновления существующих записей в таблице. Можно изменять состояния нескольких столбцов и задавать изменяемые строки с помощью where, который задает выборку изменяемых строк. Нельзя обновлять данные сразу в нескольких таблицах.

Update <Название таблицы> set <название колонки> 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>

[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 |

TRUNCATE - Удаление всех записей из таблицы, оставляя структуру таблицы нетронутой, не пишет логи, в отличие от delete, если есть ссылочное ограничение так же как и delete не позволит удалить строки.

TRUNCATE TABLE - Удаление всех записей из таблицы, посредством перемещения указателя high water mark

COMMENT: Добавление комментариев к объектам базы данных.

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