**Базы данных и SQL**

База данных (БД) – структурированное хранилище информации, организованные таким образом, чтобы обеспечить хранение, управление и доступ к информации. Чаще всего эти данные взаимосвязаны. Сама по себе БД – это просто хранилище данных, остальные свойства базы данных обеспечиваются за счет системы управления базой данных.

Система управления базой данных (СУБД) – программные средства для управления данными. Отвечает за поддержку языка БД (обеспечивает интерфейс для взаимодействия с данными), за механизм хранения и извлечения данных, за оптимизацию процессов извлечения данных.

**Виды БД:**

*Без использования СУБД*

|  |  |
| --- | --- |
| Файловые | Данные хранятся в виде текста в файле. |
| Иерархические | Имеет древовидную структуру (узел может иметь только одного родителя) Пример: файловая система. |
| Сетевые | Имеет структуру графа (узел может иметь более одного родителя) |

*С использованием СУБД*

|  |  |
| --- | --- |
| Реляционные | Структура в виде таблиц, связанных между собой |
| NoSQL | Общее название для нереляционных БД. |

**Типы СУБД:**

NoSQL (not only SQL) – Нереляционные СУБД, в которых данные могут быть организованы в виде иных структур, отличных от таблиц, ориентированных под свои задачи.

Используется язык отличный от SQL.

Отсутствие жесткой схемы данных (Следствие отсутствия схемы — эффективность работы с разреженными (sparse) данными. Если в одном документе есть поле date\_published, а во втором — нет, значит никакого пустого поля date\_published для второго создано не будет)

Отказ от атомарности и согласованности данных в пользу горизонтального масштабирования.

1) Ключ-значение – база данных организована в виде словаря. Некая таблица, имеющая уникальный ключ и соответствующее ему значение. Подходит для кэширования. [Redis]

Высокая скорость, легкое масштабирование. Не подходит для сложных структур (сущности с разными типами данных) и сложных запросов.

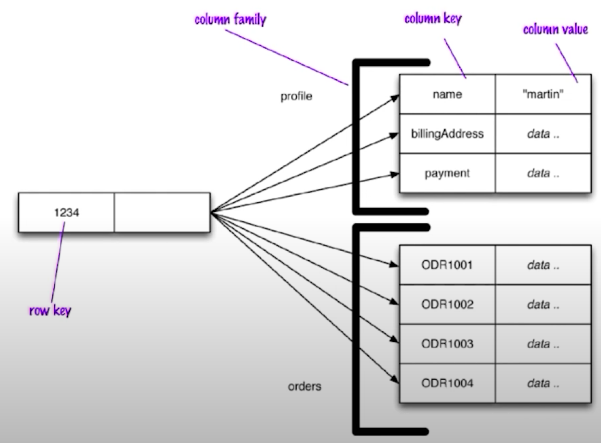
2) Колоночные – Данные организованы в виде таблиц так же, как и в реляционной СУБД, но не построчно (для чтения колонки придется вычитать всю строку до нужной колонки), а «поколоночно», т.е. колонка — это отдельная структура, в которой содержатся данные для всех строк по определенному ключу.

Подходит для аналитических запросов на больших объемах данных (эффективна при запросах для более 100 миллионов строк) – OLAP. [Apache druid]

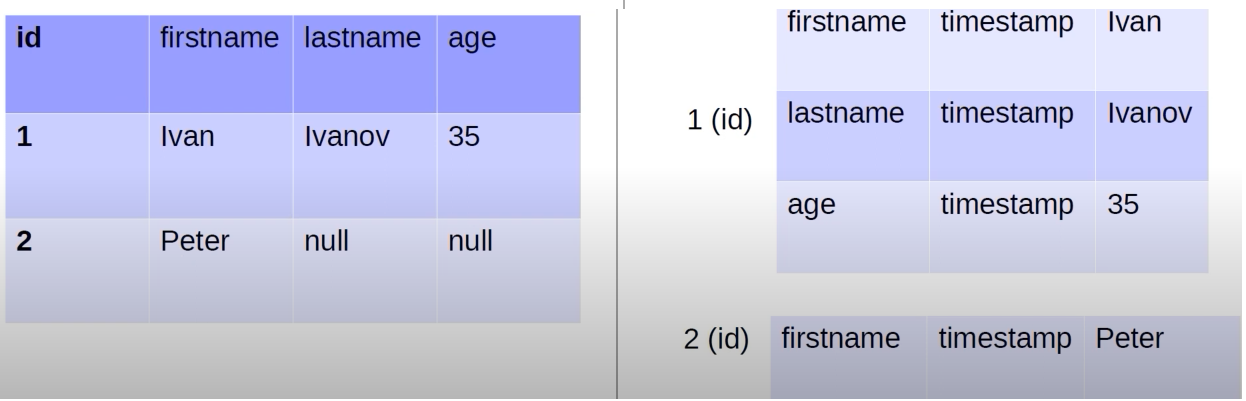
3) Ширококолоночные (Wide Column Stores) – Каждому ключу соответствует набор column family – логическая группировка колонок, каждая из которых состоит из множества сущностей ключ (названия колонки) – значение.

В отличие от реляционных не имеют жесткой схемы (в случае добавления или отсутствия значений не нужно хранить null) (см. рис.2), лучшее сжатие, горизонтально масштабируема [HBase, Cassandra]

Общая структура:



Реляционная СУБД Колоночная СУБД



4) Документо-ориентированная – Данные организованы в виде коллекций (аналог таблицы), которые в свою очередь содержат документы (аналог строки в таблице). Документ представлен парами ключ-значение, имеет формат JSON или BSON

Гибкий – не требует схем данных, данные хранятся в одной сущности – не требует соединений, как в реляционной, горизонтально масштабируемый

5) Графовые – имеют структуру графа, необходимы в ситуации, когда нужно отразить большое кол-во связей между сущностями. [Neo4j, AllegroGraph]

6) Реляционные – Данные организованы в виде отношений (таблиц) которые имеют жесткую схему организации, систему операций над которыми определяет реляционная алгебра, операции выражаются языком SQL. Согласованность данных в приоритете, заточена под транзакции, которые поддерживают свойства ACID. Построчное хранение данных – быстрая вставка, но накладные расходы при чтении данных.

**CAP теорема –** распределенная система не может в полной мере обеспечивать свойства consistency (согласованность), availability (доступность), partition tolerance (устойчивость к разделению)

Consistency – Гарантирует, что запрос к системе выдаст последнюю версию изменений, даже если изменение было произведено на другом узле – актуальность данных.

Availability - Каждый запрос завершается успешным ответом.

Partition tolerance – Система продолжает функционировать при наличии разделения/разрыва сети между узлами

CA системы – такая система подразумевает, что если выходит из строя один из компонентов системы, то она перестает работать

CP – система не гарантирует, что ответ от нее будет получен в разумные сроки (пока запрашиваемые данные не будут синхронизованы на всех узлах, клиент будет ждать ответ на запрос)

AP – система не гарантирует консистентность данных, такие системы не подразумевают того, что данные будут всегда противоречивы, они говорят о том, что данные не будут констстентны в каждый момент времени

Данная «теорема» является условной, не является теоремой в математическом понимании и содержит много противоречий [1](https://habr.com/ru/articles/231703/), [2](https://habr.com/ru/articles/322276/), [3](https://habr.com/ru/articles/328792/), [4](https://habr.com/ru/articles/258145/)

Для описания свойств SQL и NoSQL систем используются ACID и BASE

**ACID –** свойство транзакций, направленные на обеспечение надежности и целостности данных.

*Atomicity* (Атомарность) – Транзакция не может быть зафиксирована частично, то есть не может иметь промежуточных состояний, она либо выполнена полностью, либо не выполнена вовсе.

*Consistency* (Согласованность) - свойство, гарантирующее, что каждая успешная транзакция зафиксирует только допустимые результаты. То есть, это гарантия того, что при успешной транзакции будут выполнены все правила, ограничения, которые предъявляет система к конкретным данным, иначе транзакция не будет выполнена и данные в системе вернутся к прежнему состоянию.

*Isolation* (Изолированность) – Параллельные транзакции (транзакции, работающие с одними и теми же данными) не должны влиять на результаты выполнения друг друга.

Durability (Надежность) – Если клиент получил подтверждение от системы подтверждение завершения транзакции, то эти изменения не будут отменены из-за сбоя.

***Уровни изоляции для ACID -***

**BASE** – альтернативный подход к построению распределенных систем или свойства распределенных систем, в которых Consistency и Availability выполнены не в полной мере.

Basically Available (Базовая доступность) – на любой запрос будет дан ответ, но он может отражать неконсистентные данные.

Soft state (Неустойчивое состояние) – данные в системе могут меняться, даже если изменений в системе нет

Eventual consistency (согласованность в конечном итоге) – при отсутствии изменений в системе, в конечном итоге, все данные будут согласованы.

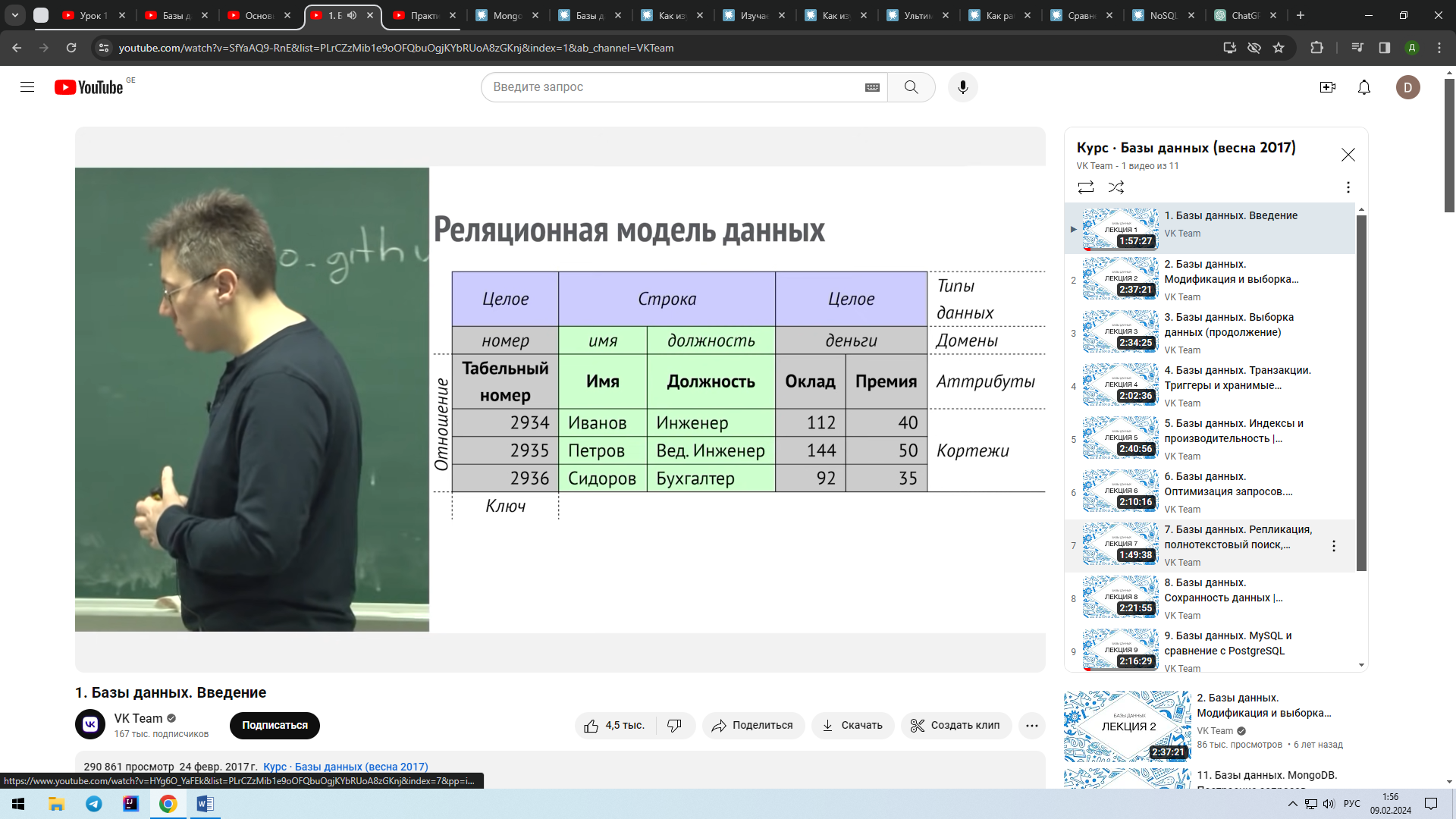
**OLAP/OLTP –** подходы для обработки данных в БД

*OLAP* **(**Online Analytical Processing**) –** Предназначен для аналитической обработки данных: анализ и агрегация больших объемов данных, заточен под чтение, а не под изменение данных.

*OLTP*(Online Transaction Processing) – Предназначен для обработки транзакций в реальном времени.

**Подробнее про реляционные СУБД**

*Реляционная модель данных:*



*Атрибут (столбец) –* Имеет Тип данных, Домен, Имя

*Домен –* некое условие для типа данных (ограничение)

В качестве первичного ключа может служить либо естественный ключ (т.е. уже существующий атрибут). Минусы – проблемы, в случае его изменения, так как нужно будет поменять его во всех таблицах, может занимать больше памяти. Либо суррогатный – к отношению добавляется новый атрибут никак его не характеризующий, например, порядковый номер или id, в качестве значения используется число с автоинкрементом.

*Строка/кортеж* – одна запись в таблице, т.е. набор значений каждого атрибута для определенного ключа

*Отношение -* таблица

*Сущность* – Объект предметной области, который может быть описан

Результирующий набор – результат запроса на языке SQL

**Ключ –** аргумент или набор аргументов, по которому(-ым) можно гарантированно идентифицировать строку. Это достигается за счет свойства уникальности и наличия значения отличного от null.

**Primary key, foreign key –**

[**Нормализация данных**](https://www.youtube.com/watch?v=zqQxWdTpSIA&ab_channel=ListenIT) **–** процессорганизации структуры хранения данных, который позволяет привести БД к минимальной избыточности (повторяемые данные). Избыточность, как правило, устраняется за счет декомпозиции отношений (разбиение одной таблицы на несколько). Направлена на устранение аномалий путем устранения избыточного дублирования и повышение производительности.

Избыточность данных создает предпосылки для возникновения аномалий (некоторые примеры):

1) Дублирующая информация – ненормализованная таблица может содержать множество одинаковых значений одного типа, что увеличивает объем данных для хранения

2) Изменение значения типа - Если значение типа изменится, то нужно изменить все строки, содержащие это значение

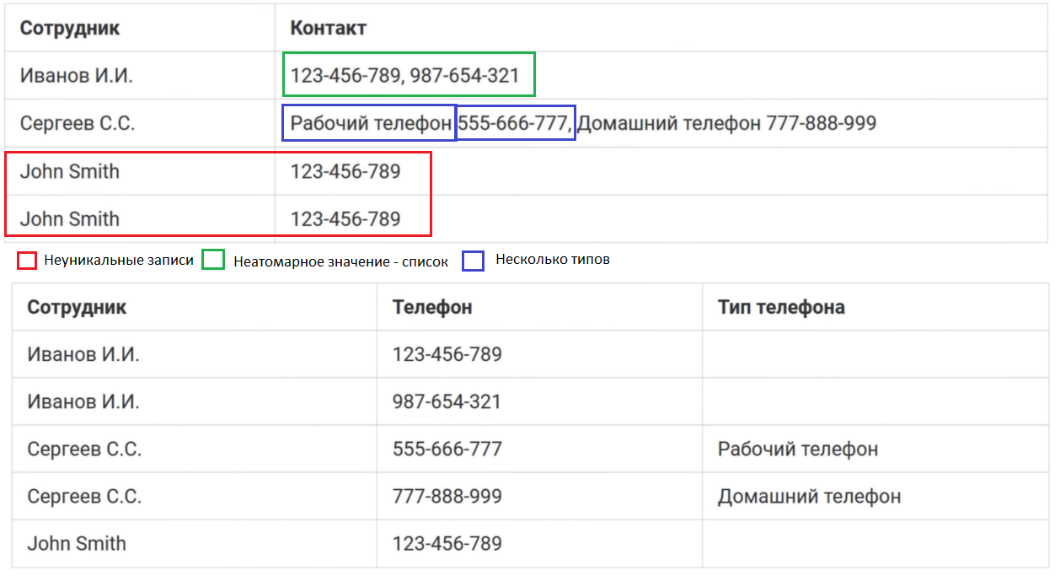
3) Потеря данных - При заполнении таблицы данными типа он может быть записан неверно, что приведет к невозможности включения этих данных в нужный тип.

4) Потеря типа - Информация о типах хранится в таблице представляющей сущности. Если тип представлен одной сущностью в таблице, то удаление этой сущности приведет к потере типа.

***Нормальные формы*** *–* Набор правил и критериев, которым должна отвечать БД для соответствия той или иной форме БД. Каждая последующая форма подразумевает, что БД должна соответствовать всем предыдущим нормальным формам.

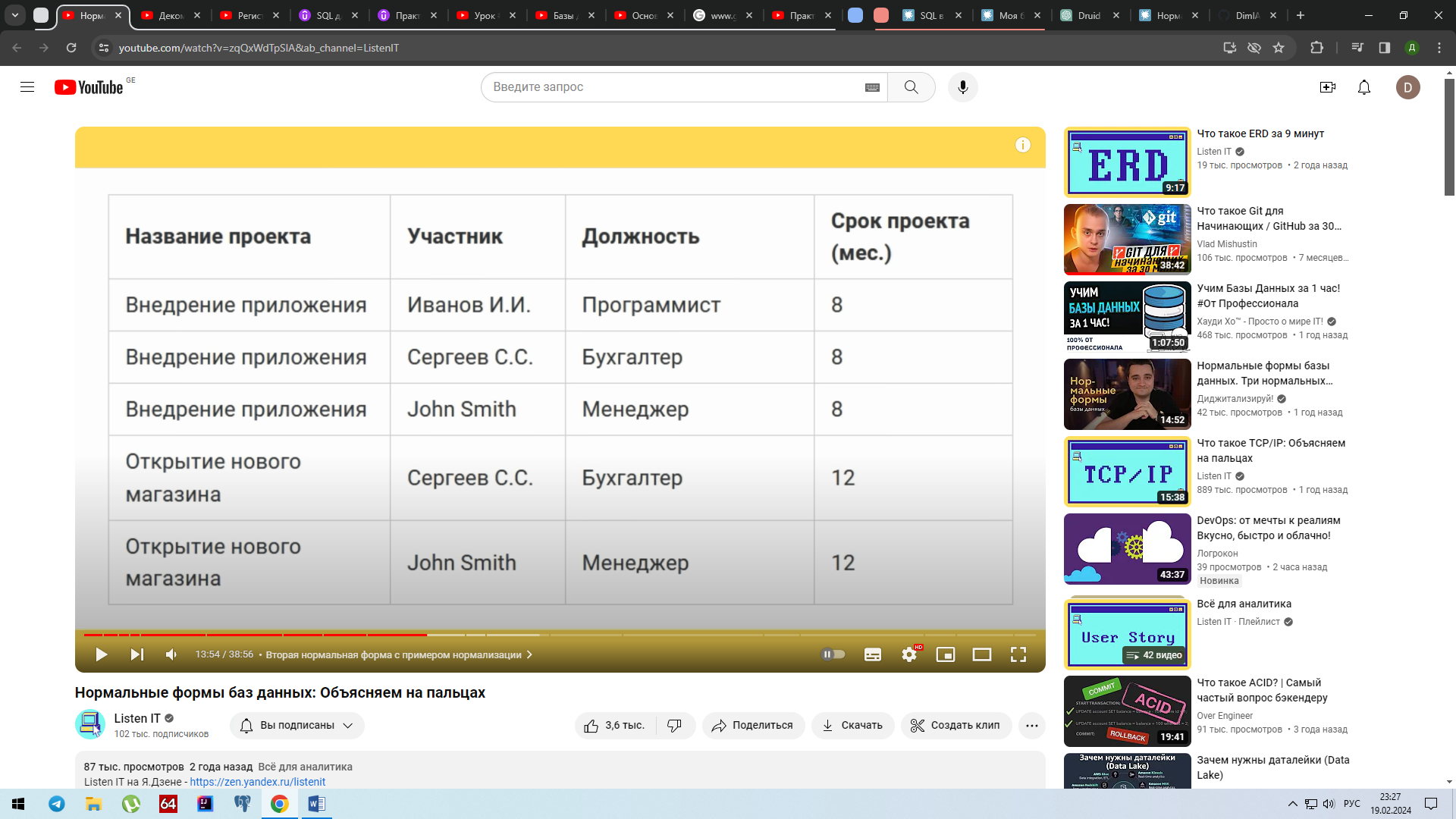
0НФ – Приведение БД к виду, соответствующему базовым принципам реляционной теории. Строки в таблицах не должны быть пронумерованы (порядок строк и столбцов не имеет значения).

*1НФ –* Строка – уникальна (нет полностью идентичных строк), ячейка - атомарна (хранит одно неделимое по смыслу значение - не должно быть массивов и списков значений), столбец – хранит данные только одного типа.



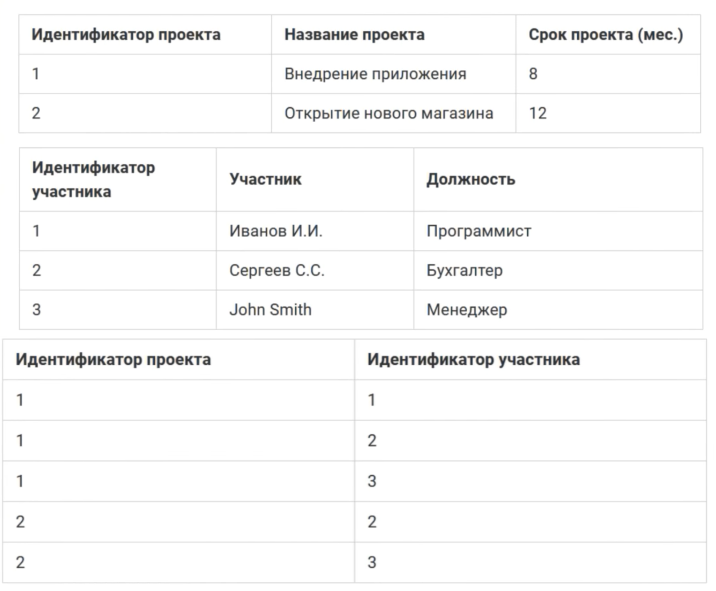
*2НФ* – Таблица должна иметь первичный ключ, для возможности однозначно идентифицировать каждый кортеж (ключ может быть простой или составной). В случае, если необходим составной ключ, то все неключевые атрибуты таблицы должны зависеть от полного ключа, т.е. невозможно однозначно определить запись в таблице, зная только часть ключа.

Если таблица выражена отношением многие ко многим, то для однозначной идентификации записи потребуется использование составного ключа. То есть, имея только название проекта, невозможно однозначно определить строку. Поэтому нужно выбрать несколько атрибутов, которые будут обеспечивать свойство уникальности (название проекта, участник).



Для обеспечения второго свойства, необходимо проверить, можно ли получить должность или срок проекта, зная только часть ключа. В данном случае, зная только название можно определить срок, а зная участника – определить должность – второе свойство не выполняется. Значит нужно прибегнуть к декомпозиции.

Объединить связанные типы в сущности – участник и должность, проект и срок. И присвоить им простые первичные ключи, так же создать третью отношение, отражающее связь проектов и участников.

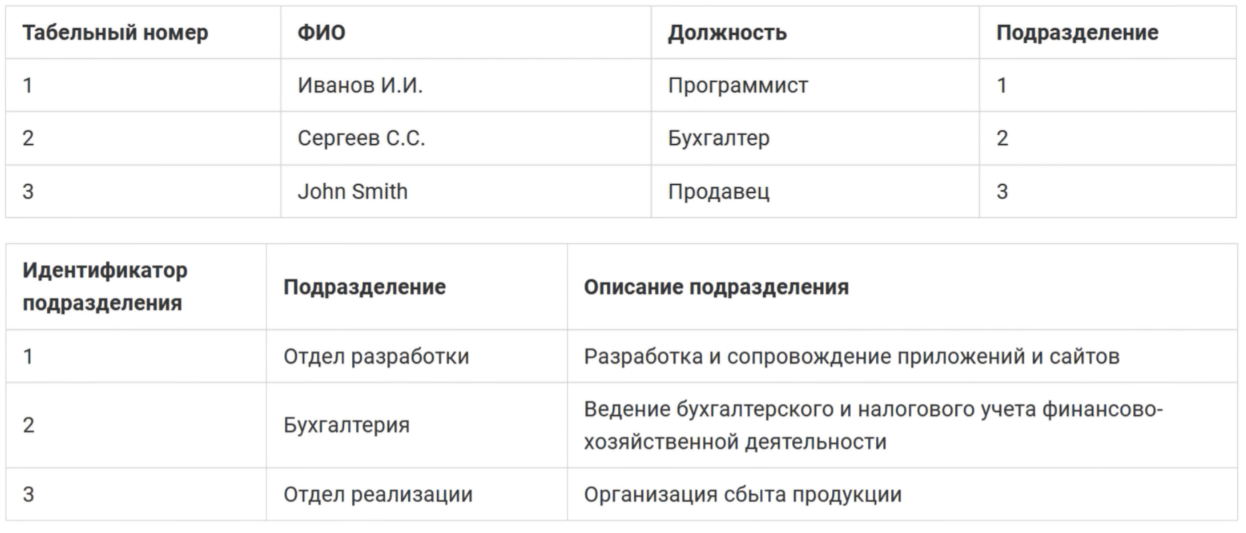


*3НФ* – В таблице должна отсутствовать транзитивная (непрямая) зависимость. То есть неключевые атрибуты напрямую зависят только от первичного ключа, но не от других атрибутов. Неключевые столбцы не должны играть роль ключа.

Описание подразделения относится к подразделению, а не к первичному ключу, то есть подразделение (неключевой атрибут) может играть роль первичного ключа.

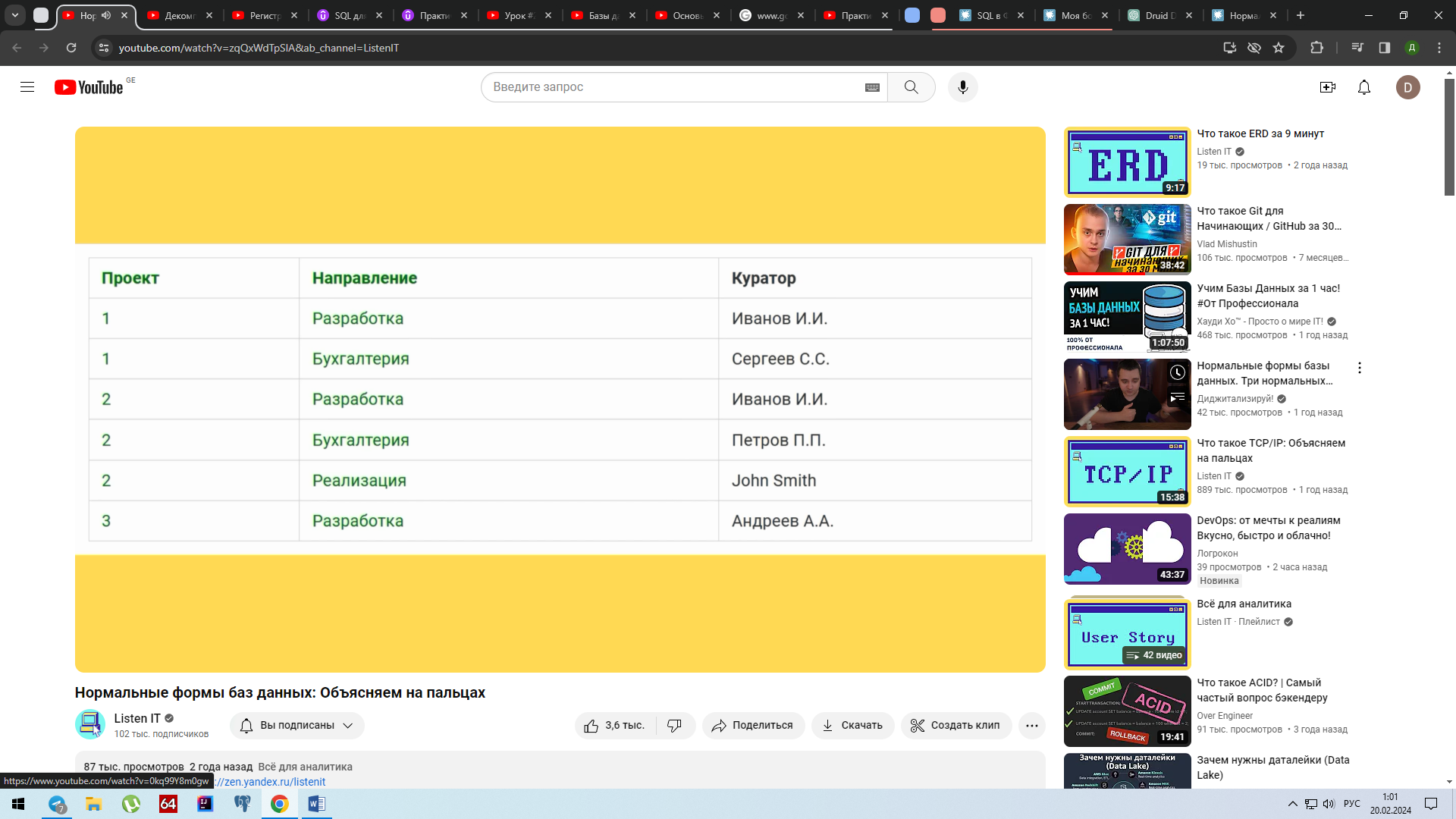


Для приведения к 3НФ – декомпозиция и создание связи через внешние ключи:



Нормальная форма Бойса-Кодда (усиленная 3НФ) – Ключевые атрибуты составного ключа не должны зависеть от неключевых атрибутов. То есть данная форма актуальна только для отношений, имеющих составной первичный ключ.

Выбрав неключевой атрибут (куратора) можно точно определить часть составного ключевого атрибута (направление)



Решается также декомпозицией



Все последующие формы нормализации практически не встречаются.

*4НФ* – в таблице не должно быть многозначных зависимостей. Наличие таких зависимостей приводит невозможности редактирования одной зависимости без вмешательства в другую

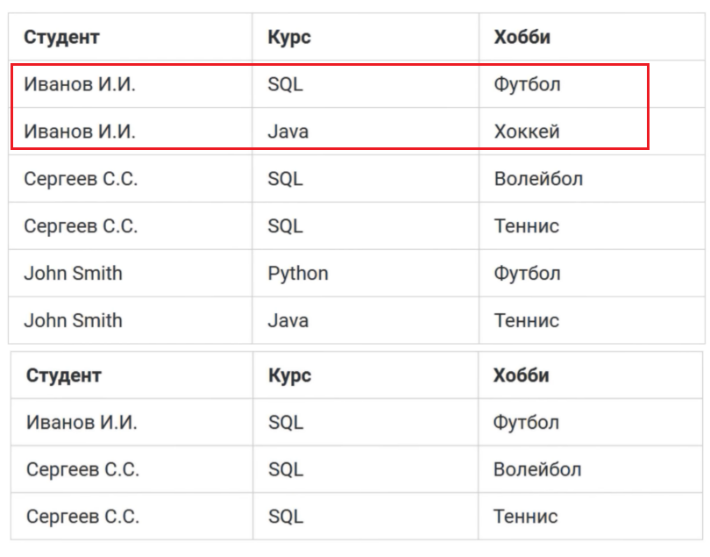
Пример многозначной зависимости – Есть аргументы A, B, C. B и C не связаны между собой, но связаны по отдельности зависят от A. То есть для каждого A есть множество значений B и множество значений C.

Таблица с расписанием курсов – Преподаватель и Аудитория между собой независимы, но по отдельности каждый из них зависит от типа Курс.



Удаление преподавателя Иванов приведет к потере типа Аудитория – 101 и 203;

Так же при составлении расписания аудитория может быть известна, а преподаватель не определен, в таком случае тип преподаватель будет равен null или default, что так же является аномалией.



Список хобби студентов, которые посещают курс SQL выводит неверный результат, так как Иванов посещает курс SQL и имеет хобби Футбол и Хоккей, чего нет в результирующей выборке.

*5НФ* – Таблица соответствует 5НФ, если возможна ее декомпозиция без потерь - разбиение таблицы на несколько и возможность их последующего соединения в точно такую же таблицу (без потерь строк и возникновения новых).

Таблица будет находиться в 5НФ, если она не содержит зависимостей соединения – существование связи между двумя атрибутами, через третий атрибут, который не является ключом

Невозможно точно определить, находится ли таблица в 5НФ, точно не зная предметной области и всех возможных зависимостей.

*Так же есть доменно-ключевая и 6 нормальные формы.*

Минусы сильной нормализации – теряется понятность данных в таблице и нужно писать более сложные запросы к таблице, снижение производительности СУБД – до 4НФ производительность растет, если использовать 4НФ и последующие, то у такой БД будет проседать производительность, но при этом уменьшаться кол-во аномалий.

Исходя из этого, нужно искать баланс между удобством работы, ее производительностью и допустимым кол-вом аномалий.

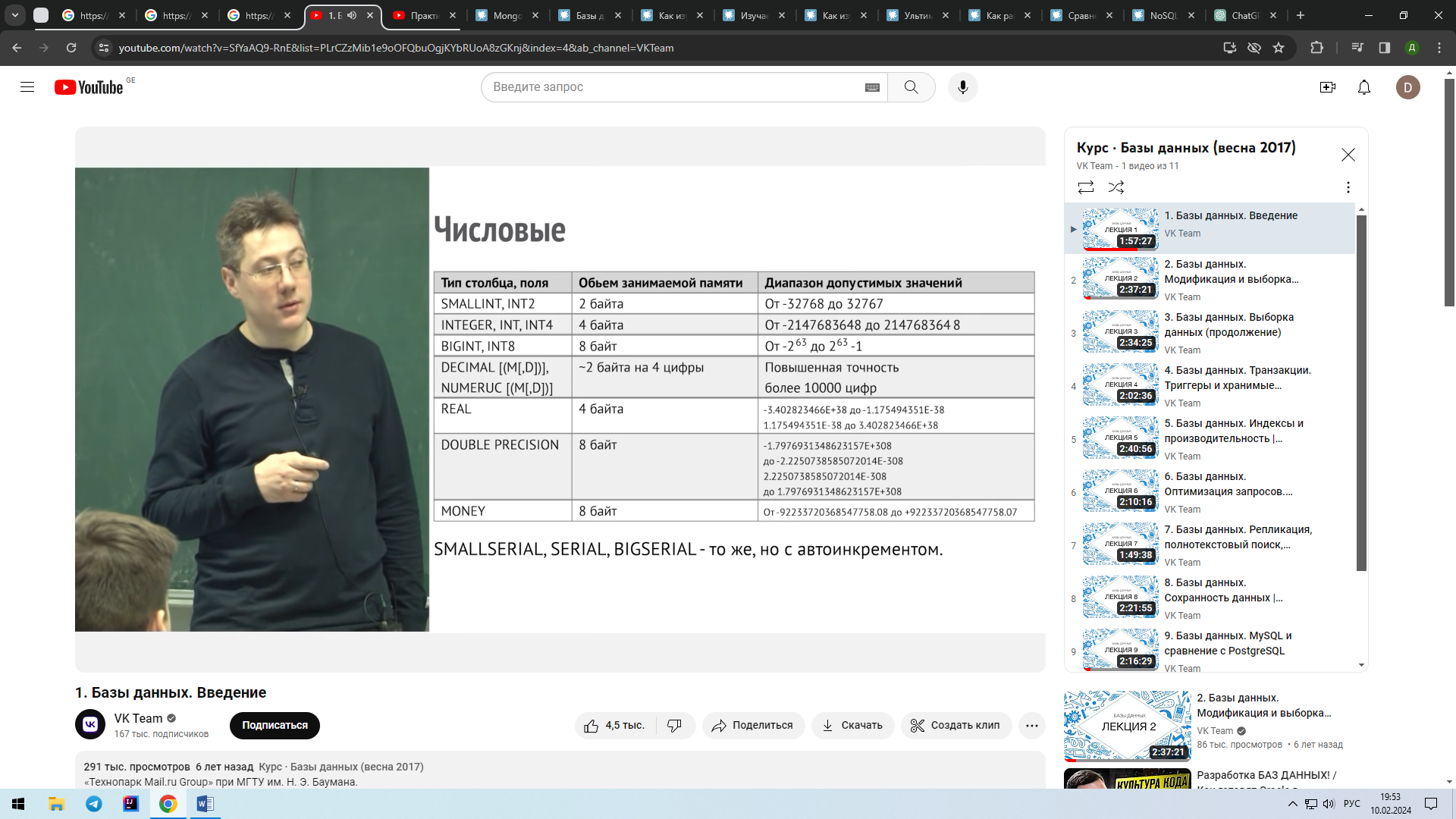
*Денормализация* - пренебрежение формами нормализации для увеличения производительности и упрощения написания запросов.

**Типы связей**

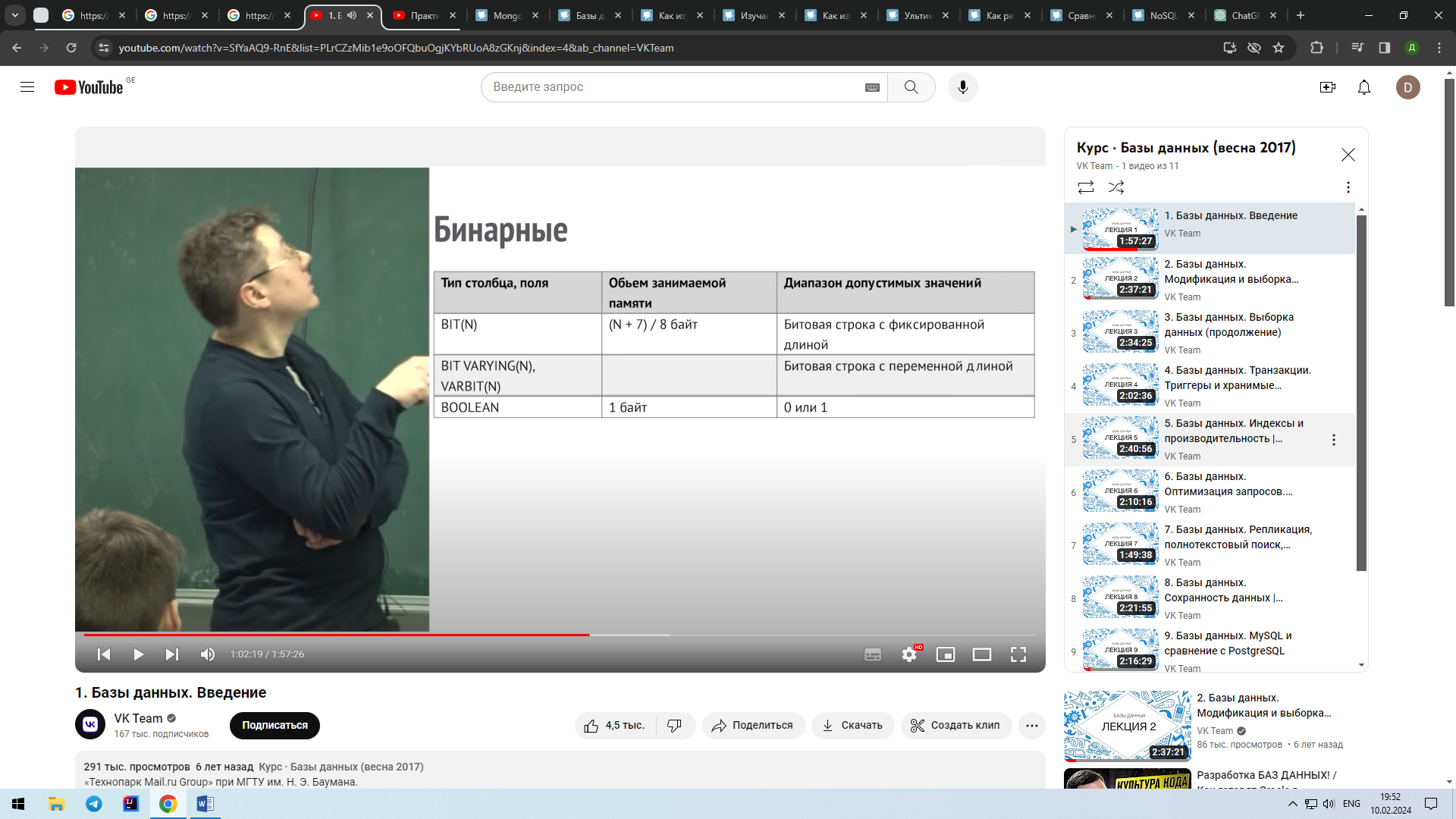
**Constraints**

**Типы данных в PostgreSQL**

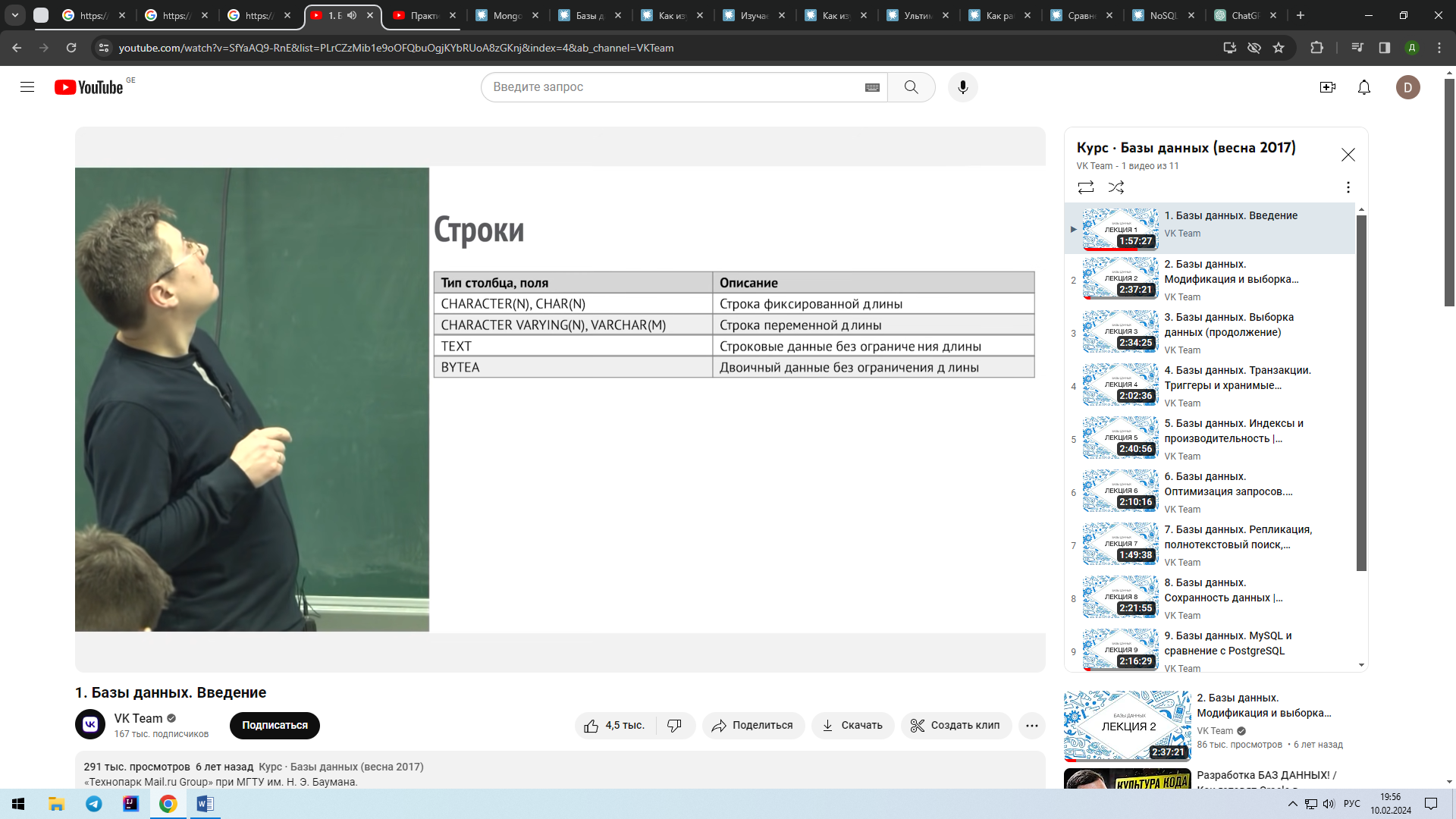
*Числовые*

*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

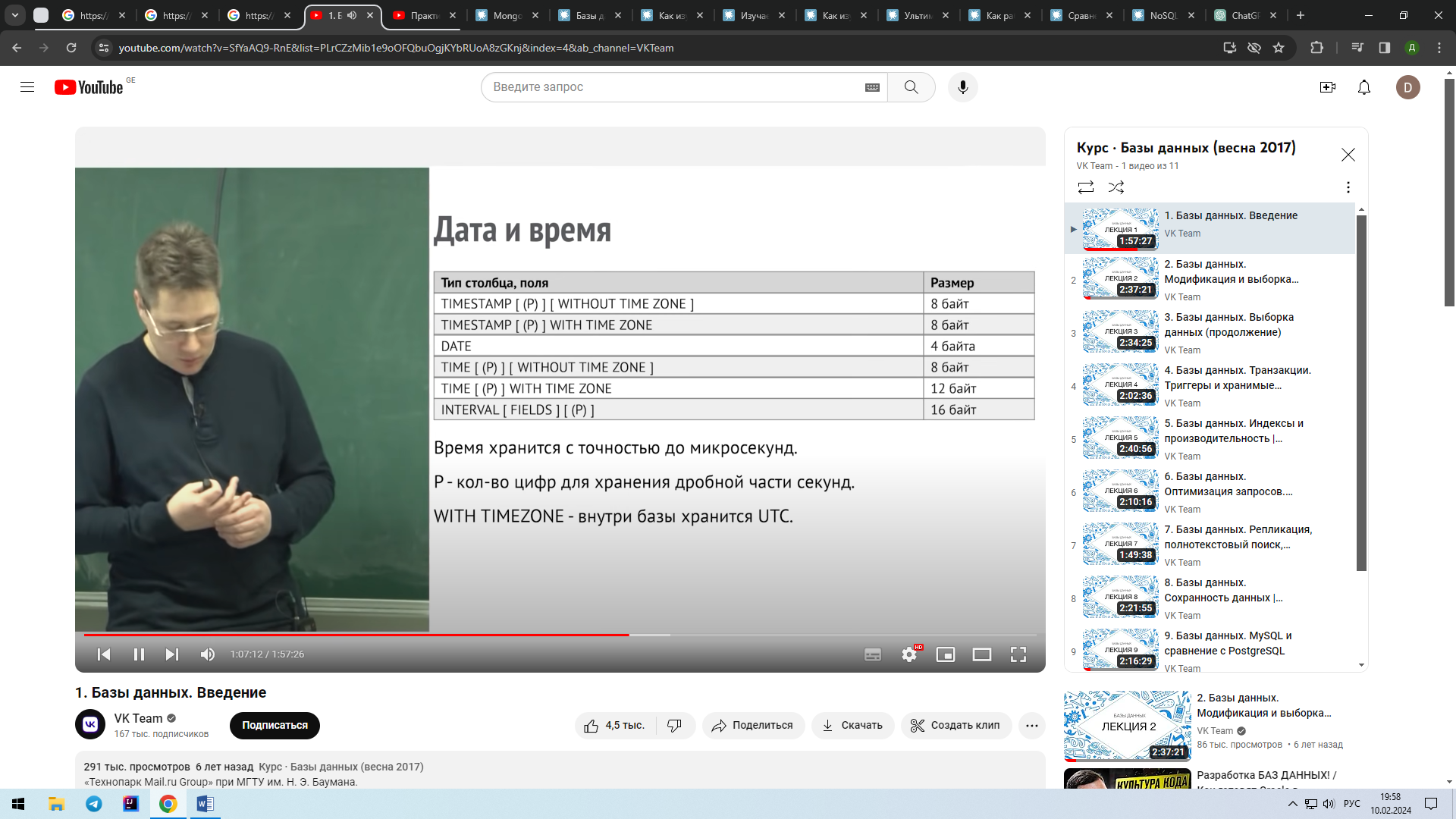


Bit(n) – фиксированная длина, если меньше – дописываются нули.

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 + миллесекунды

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.

Общий вид команды:

Select <Имя столбца> from <Имя таблицы> where<условия> order by;

*Distinct* – убирает дубликаты из результирующего набора строк.

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)

SELECT \* FROM students AS name\_alias; - для переименования таблицы.

SELECT id AS student\_id, name AS student\_name FROM students; - для переименования колонок

Оператор as можно опускать.

*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

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). Является альтернативой функциям NVL, NVL2, NVLIF в Oracle | |
| 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. |
| *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>

\*Подзапросы, RETURNING, FOR UPDATE или FOR SHARE\*\*\*

**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)

**Merge\*\*\***

**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.

**Select for update** – Блокировка на уровне строки, исключение конкурентного доступа\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**NameSpace\*\*\*\***

**DDL** ([back](#Категории))

1. **CREATE:** Создание объектов базы данных, таких как таблиц, индексов, представлений и других.
   * **CREATE TABLE**: Создание таблицы.
   * **CREATE INDEX**: Создание индекса.
   * **CREATE VIEW**: Создание представления.
   * **CREATE DATABASE**: Создание новой базы данных.
   * **Create Sequence**
   * **Create Schema**
   * **Create Trigger**
   * **Create Function**

Сreate table schema.table Organization Heap – вроде как не используется

(Название колонки, тип колонки, default значение)

1. **ALTER:** Изменение структуры существующих объектов.
   * **ALTER TABLE**: Изменение структуры таблицы (например, добавление/удаление столбцов, изменение типов данных).
   * **ALTER INDEX**: Изменение индекса.
   * **ALTER VIEW**: Изменение представления.
   * **ALTER DATABASE**: Изменение параметров базы данных.
2. **DROP:** Удаление объектов базы данных.
   * **DROP TABLE**: Удаление таблицы.
   * **DROP INDEX**: Удаление индекса.
   * **DROP VIEW**: Удаление представления.
   * **DROP DATABASE**: Удаление базы данных.

Drop table schema.table;

Не получится удалить, если есть активные сессии с этой таблицей или есть зависимости, для удаления таблицы вместе с зависимостями Cascade

1. **TRUNCATE:** Удаление всех записей из таблицы, оставляя структуру таблицы нетронутой.
   * **TRUNCATE TABLE**: Удаление всех записей из таблицы.
2. **COMMENT:** Добавление комментариев к объектам базы данных.
   * **COMMENT ON TABLE**: Добавление комментария к таблице.
   * **COMMENT ON COLUMN**: Добавление комментария к столбцу таблицы.

Перемещает указатель high water mark

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*С функциями, подзапросы, в несколько таблиц, Constraint и его виды\*

Дефолт значения для столбцов, в случае отсутствия значения при добавлении/обновлении

Using

:: - знак приведения типов

’12:34:56’ :: Time – результат тип данных time

\*Подзапросы\*