Оглавление

SQL	
1. Что такое DDL? Какие операции в него входят? Рассказать про них.	DDL DDL-это краткое название языка определения данных, который имеет дело со схемами баз данных и описаниями того, как данные должны храниться в базе данных. CREATE – для создания базы данных и ее объектов, таких как (таблица, индекс, представления, процедура хранения, функция и триггеры). ALTER – изменяет структуру существующей базы данных. DROP – удаление объектов из базы данных. TRUNCATE – удалить все записи из таблицы; кроме того, удаляются все места, выделенные для записей.
2. Что такое DML? Какие операции в него входят? Рассказать про них.	операторы манипуляции данными (Data Manipulation Language, DML): SELECT выбирает данные, удовлетворяющие заданным условиям, INSERT добавляет новые данные, UPDATE изменяет существующие данные, DELETE удаляет данные при выполнении условия WHERE;
3. Что такое TCL? Какие операции в него входят? Рассказать про них.	операторы управления транзакциями (Transaction Control Language, TCL): BEGIN служит для определения начала транзакции COMMIT применяет транзакцию, ROLLBACK откатывает все изменения, сделанные в контексте текущей транзакции, SAVEPOINT разбивает транзакцию на более мелкие.
4. Что такое DCL? Какие операции в него входят? Рассказать про них.	операторы определения доступа к данным (Data Control Language, DCL): GRANT предоставляет пользователю (группе) разрешения на определенные операции с объектом, REVOKE отзывает ранее выданные разрешения, DENY задает запрет, имеющий приоритет над разрешением;
5. Нюансы работы с NULL в SQL. Как проверить поле на NULL?	NULL - **специальное значение** (псевдозначение), которое может быть записано в поле таблицы базы данных. NULL соответствует понятию «пустое поле», то есть «поле, не содержащее никакого значения». NULL **означает отсутствие**, неизвестность **информации**. Значение NULL не является значением в полном смысле слова: по определению оно означает отсутствие значения и не принадлежит ни одному типу данных. Поэтому NULL не равно ни логическому значению FALSE, ни пустой строке, ни 0. **При сравнении NULL с любым значением будет получен результат NULL**, а не FALSE и не 0. Более того, **NULL не равно NULL**! команды: IS NULL, IS NOT NULL
6. Виды Join'ов?	JOIN - оператор языка SQL, который является реализацией операции соединения реляционной алгебры. Предназначен для обеспечения выборки данных из двух таблиц и включения этих данных в один результирующий набор. Особенностями операции соединения являются следующее: - в схему таблицы-результата входят столбцы обеих исходных таблиц (таблиц-операндов), то есть схема результата является «сцеплением» схем операндов;

	- каждая строка таблицы-результата является «сцеплением» строки из одной таблицы-операнда со строкой второй таблицы-операнда; - при необходимости соединения не двух, а нескольких таблиц, операция соединения применяется несколько раз (последовательно). **Какие существуют типы JOIN?** (INNER) JOIN Результатом объединения таблиц являются записи, общие для левой и правой таблиц. Порядок таблиц для оператора не важен, поскольку оператор является симметричным. LEFT (OUTER) JOIN Кортежи из внутреннего соединения, и не вошедшие во внутреннее соединение кортежи из левого источника. Атрибуты в кортежах, которые не имеют совпадений по общим столбцам заполняются неопределенными значениями. Порядок таблиц для оператора важен, поскольку оператор не является симметричным. RIGHT (OUTER) JOIN FULL (OUTER) JOIN Результатом объединения таблиц являются все записи, которые присутствуют в таблицах. Порядок таблиц для оператора не важен, поскольку оператор является симметричным. CROSS JOIN (декартово произведение) SELECT * FROM actor NATURAL JOIN film_actor NATURAL JOIN film Обратите внимание, в этом запросе нет необходимости указывать какие-либо критерии объединения, поскольку предложение NATURAL JOIN автоматически определяет столбцы, имеющие одинаковые имена в обеих объединяемых таблицах, и помещает их в «скрытое» предложение USING. Если первичные и внешние ключи имеют одинаковые имена, этот подход может показаться полезным, однако это не так.

7. Что лучше использовать join или подзапросы? Почему?	Обычно лучше использовать JOIN, поскольку в большинстве случаев он более понятен и лучше оптимизируется СУБД (но 100% этого гарантировать нельзя). Так же JOIN имеет заметное преимущество над подзапросами в случае, когда список выбора SELECT содержит столбцы более чем из одной таблицы. Подзапросы лучше использовать в случаях, когда нужно вычислять агрегатные значения и использовать их для сравнений во внешних запросах.
8. Что делает UNION?	В языке SQL ключевое слово UNION применяется для объединения результатов двух SQL-запросов в единую таблицу, состоящую из схожих записей. Оба запроса должны возвращать одинаковое число столбцов и совместимые типы данных в соответствующих столбцах. Необходимо отметить, что UNION сам по себе не гарантирует порядок записей. Записи из второго запроса могут оказаться в начале, в конце или вообще перемешаться с записями из первого запроса. В случаях, когда требуется определенный порядок, необходимо использовать ORDER BY. Разница между UNION и UNION ALL заключается в том, что UNION будет пропускать дубликаты записей, тогда как UNION ALL будет включать дубликаты записей.
9. Чем WHERE отличается от HAVING (отв то что используются в разных частях запроса - недостаточно)?	WHERE нельзя использовать с агрегатными функциями, HAVING можно (предикаты тоже). В HAVING можно использовать псевдонимы только если они используются для наименования результата агрегатной функции, в WHERE можно всегда. <------????????? HAVING стоит после GROUP BY, но может использоваться и без него. При отсутствии предложения GROUP BY агрегатные функции применяются ко всему выходному набору строк запроса, т.е. в результате мы получим всего одну строку, если выходной набор не пуст.
10. Что такое ORDER BY?	ORDER BY упорядочивает вывод запроса согласно значениям в том или ином количестве выбранных столбцов. Многочисленные столбцы упорядочиваются один внутри другого. Возможно определять возрастание ASC или убывание DESC для каждого столбца. По умолчанию установлено - возрастание.
11. Что такое DISTINCT?	DISTINCT указывает, что для вычислений используются только уникальные значения столбца.
12. Что такое GROUP BY?	GROUP BY используется для агрегации записей результата по заданным атрибутам. Создает отдельную группу для всех возможных значений (включая значение NULL) При использовании GROUP BY все значения NULL считаются равными.
13. Что такое LIMIT?	Ограничивает выборку заданным числом.
14. Что такое EXISTS?	EXISTS берет подзапрос, как аргумент, и оценивает его как TRUE, если подзапрос возвращает какие-либо записи и FALSE, если нет.
15. Расск. про операторы IN, BETWEEN, LIKE.	• IN - определяет набор значений. SELECT * FROM Persons WHERE name IN ('Ivan','Petr','Pavel'); • BETWEEN определяет диапазон значений. В отличие от IN, BETWEEN чувствителен к порядку, и первое значение в предложении должно быть первым по алфавитному или числовому порядку. SELECT * FROM Persons WHERE age BETWEEN 20 AND 25; • LIKE применим только к полям типа CHAR или VARCHAR, с которыми он используется чтобы находить подстроки. В качестве условия используются символы шаблонизации (wildkards) - специальные символы, которые могут соответствовать чему-нибудь: _ замещает любой одиночный символ. Например, 'b_t' будет соответствовать словам 'bat' или 'bit', но не будет соответствовать 'brat'. % замещает последовательность любого числа символов. Например '%p%t' будет соответствовать словам 'put', 'posit', или 'opt', но не 'spite'. SELECT * FROM UNIVERSITY WHERE NAME LIKE '%o';
16. Что делает оператор	MERGE позволяет осуществить слияние данных одной таблицы с данными другой таблицы. При слиянии таблиц проверяется условие, и если оно истинно, то выполняется UPDATE, а если нет - INSERT. При этом

MERGE? Какие у него есть ограничения ?	изменять поля таблицы в секции UPDATE, по которым идет связывание двух таблиц, нельзя. **MERGE Ships AS t** -- таблица, которая будет меняться **USING (SELECT запрос) AS s ON (t.name = s.ship)** -- условие слияния **THEN UPDATE SET t.launched = s.year** -- обновление **WHEN NOT MATCHED** -- если условие не выполняется **THEN INSERT VALUES(s.ship, s.year)** -- вставка
17. Какие агрегатные функции вы знаете?	Агрегатных функции - функции, которые берут группы значений и сводят их к одиночному значению. Несколько агрегатных функций: COUNT - производит подсчет записей, удовлетворяющих условию запроса; CONCAT - соединяет строки; SUM - вычисляет арифметическую сумму всех значений колонки; AVG - вычисляет среднее арифметическое всех значений; MAX - определяет наибольшее из всех выбранных значений; MIN - определяет наименьшее из всех выбранных значений.
18. Что такое ограничения (constraints)? Какие вы знаете?	Ограничения - это ключевае слова, которые помогают установить правила размещения данных в базе. Используются при создании БД. **NOT NULL** указывает, что значение не может быть пустым. **UNIQUE** обеспечивает отсутствие дубликатов. **PRIMARY KEY** - комбинация NOT NULL и UNIQUE. Помечает каждую запись в базе данных уникальным значением. **CHECK** проверяет вписывается ли значение в заданный диапазон (s_id int CHECK(s_id > 0)) **FOREIGN KEY** создает связь между двумя таблицами и защищает от действий, которые могут нарушить связи между таблицами. FOREIGN KEY в одной таблице указывает на PRIMARY KEY в другой. **DEFAULT** устанавливает значение по умолчанию, если значения не предоствлено (name VARCHAR(20) DEFAULT 'noname'). Какие отличия между PRIMARY и UNIQUE? По умолчанию PRIMARY создает кластерный индекс на столбце, а UNIQUE - некластерный. PRIMARY не разрешает NULL записей, в то время как UNIQUE разрешает одну (а в некоторых СУБД несколько) NULL запись. Таблица может иметь один PRIMARY KEY и много UNIQUE. Может ли значение в столбце, на который наложено ограничение FOREIGN KEY, равняться NULL? Может, если на данный столбец не наложено ограничение NOT NULL.
19. Что такое суррогатные ключи?	Суррога́тный ключ — это дополнительное служебное поле, автоматически добавленное к уже имеющимся информационным полям таблицы, предназначение которого — служить первичным ключом.
20. Что такое индексы? Какие они бывают?	Индексы относятся к методу настройки производительности, позволяющему быстрее извлекать записи из таблицы. Индекс создает структуру для индексируемого поля. Необходимо просто добавить указатель индекса в таблицу. Есть три типа индексов, а именно: **Уникальный** индекс (Unique Index): этот индекс не позволяет полю иметь

	повторяющиеся значения. Если первичный ключ определен, уникальный индекс применен автоматически. **Кластеризованный** индекс (Clustered Index): сортируют и хранят строки данных в таблицах или представлениях на основе их ключевых значений. Это ускоряет операции чтения из БД. **Некластеризованный** индекс (Non-Clustered Index): внутри таблицы есть упорядоченный список, содержащий значения ключа некластеризованного индекса и указатель на строку данных, содержащую значение ключа. Каждый новый индекс увеличивает время, необходимое для создания новых записей из-за упорядочивания. Каждая таблица может иметь много некластеризованных индексов.
	Как создать индекс? b3 Индекс можно создать либо с помощью выражения CREATE INDEX: CREATE INDEX index_name ON table_name (column_name) либо указав ограничение целостности в виде уникального UNIQUE или первичного PRIMARY ключа в операторе создания таблицы CREATE TABLE.
	Имеет ли смысл индексировать данные, имеющие небольшое количество возможных значений? Примерное правило, которым можно руководствоваться при создании индекса - если объем информации (в байтах) НЕ удовлетворяющей условию выборки меньше, чем размер индекса (в байтах) по данному условию выборки, то в общем случае оптимизация приведет к замедлению выборки.
	Когда полное сканирование набора данных выгоднее доступа по индексу? Полное сканирование производится многоблочным чтением. Сканирование по индексу - одноблочным. Также, при доступе по индексу сначала идет сканирование самого индекса, а затем чтение блоков из набора данных. Число блоков, которые надо при этом прочитать из набора зависит от фактора кластеризации. Если суммарная стоимость всех необходимых одноблочных чтений больше стоимости полного сканирования многоблочным чтением, то полное сканирование выгоднее и оно выбирается оптимизатором. Таким образом, полное сканирование выбирается при слабой селективности предикатов зароса и/или слабой кластеризации данных, либо в случае очень маленьких наборов данных.

21. Чем TRUNCATE отличается от DELETE?	DELETE - оператор DML, удаляет записи из таблицы, которые удовлетворяют условиям WHERE. Медленнее, чем TRUNCATE. Есть возможность восстановить данные. TRUNCATE - DDL оператор, удаляет все строки из таблицы. Нет возможность восстановить данные - сделать ROLLBACK.
22. Что такое хранимые процедуры? Для чего они нужны?	**Хранимая процедура — объект базы данных, представляющий собой набор SQL-инструкций, который хранится на сервере.** Хранимые процедуры очень похожи на обыкновенные методы языков высокого уровня, у них могут быть входные и выходные параметры и локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных (как DDL, так и DML). Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления процессом исполнения. Хранимые процедуры позволяют повысить производительность, расширяют возможности программирования и поддерживают функции безопасности данных. В большинстве СУБД при первом запуске хранимой процедуры она компилируется (выполняется синтаксический анализ и генерируется план доступа к данным) и в дальнейшем её обработка осуществляется быстрее.
23. Что такое представления (VIEW)? Для чего они нужны?	View - виртуальная таблица, представляющая данные одной или более таблиц альтернативным образом. В действительности представление – всего лишь результат выполнения оператора SELECT, который хранится в структуре памяти, напоминающей SQL таблицу. Они работают в запросах и операторах DML точно также как и основные таблицы, но не содержат никаких собственных данных. Представления значительно расширяют возможности управления данными. Это способ дать публичный доступ к некоторой (но не всей) информации в таблице. Представления могут основываться как на таблицах, так и на других представлениях, т.е. могут быть вложенными (до 32 уровней вложенности).
24. Что такое временные таблицы? Для чего они нужны?	Подобные таблицы удобны для каких-то временных промежуточных выборок из нескольких таблиц. Создание временной таблицы начинается со знака решетки #. Если используется один знак #, то создается локальная таблица, которая доступна в течение текущей сессии. Ели используются два знака ##, то создается глобальная временная таблица. В отличие от локальной глобальная временная таблица доступна всем открытым сессиям базы данных. CREATE TABLE #ProductSummary (ProdId INT IDENTITY, ProdName NVARCHAR(20), Price MONEY)
25. Что такое транзакции? Расскажите про принципы ACID.	**Транзакция** - это воздействие на базу данных, переводящее её из одного целостного состояния в другое и выражаемое в изменении данных, хранящихся в базе данных. ACID-принципы транзакций: • **Атомарность** (atomicity) гарантирует, что транзакция будет полностью выполнена или потерпит неудачу, где транзакция представляет одну логическую операцию данных. Это означает, что при сбое одной части любой транзакции происходит сбой всей транзакции и состояние базы данных остается неизменным. • **Согласованность** (consistency). Транзакция, достигающая своего завершения и фиксирующая свои результаты, сохраняет согласованность базы данных • **Изолированность** (isolation). Во время выполнения транзакции параллельные транзакции не должны оказывать влияние на ее результат. • **Долговечность** (durability). Независимо от проблем (к примеру, потеря

	питания, сбой или ошибки любого рода) изменения, сделанные успешно завершённой транзакцией, должны остаться сохраненными после возвращения системы в работу.
26. Расск. про уровни изолированности транзакций.	В порядке увеличения изолированности транзакций и, соответственно, надежности работы с данными: • **Чтение неподтверждённых данных (read uncommitted)** — чтение незафиксированных изменений как своей транзакции, так и параллельных транзакций. Нет гарантии, что данные, измененные другими транзакциями, не будут в любой момент изменены в результате их отката, поэтому такое чтение является потенциальным источником ошибок. Возможны неповторяемое чтение, фантомы и грязное чтение. • **Чтение подтвержденных данных (read committed)** — чтение всех изменений своей транзакции и зафиксированных изменений параллельных транзакций, но процессы-писатели могут изменять уже прочитанные читателем данные. Возможны неповторяемое чтение и фантомы. • **Повторяемость чтения (repeatable read)** — Уровень, позволяющий предотвратить неповторяемое чтение. Т.е. мы не видим в исполняющейся транзакции измененные и удаленные записи этой же или другой транзакцией. Но все еще видим вставленные записи из другой транзакции. В MySQL и PostgreSQL отсутствует эффект чтения фантомов для этого уровня. • **Упорядочиваемость (serializable)** — гарантирует неизменяемость данных другими процессами до завершения транзакции. Проблемы синхронизации не возникают. https://habr.com/ru/post/469415/ https://www.youtube.com/watch?v=5Z2iFX3OeTo&ab_channel=%D0%A3%D1%80%D0%BE%B8JavaD0%BA%D0%
	При параллельном выполнении транзакций возможны следующие проблемы: Потерянное обновление (lost update) — при одновременном изменении одного блока данных разными транзакциями одно из изменений теряется; «Грязное» чтение (dirty read) — чтение данных, добавленных или измененных транзакцией, которая впоследствии не подтвердится (откатится); Неповторяющееся чтение (non-repeatable read) — при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются измененными; Фантомное чтение (phantom reads) — одна транзакция в ходе своего выполнения несколько раз выбирает множество записей по одним и тем же критериям. Другая транзакция в интервалах между этими выборками добавляет или удаляет записи или изменяет столбцы некоторых записей, используемых в критериях выборки первой транзакции, и успешно заканчивается. В результате получится, что одни и те же выборки в первой транзакции дают разные множества записей.

27. Что такое нормализация и денормализация? Расскажите про 3 нормальные формы?в	**Нормализация - это процесс преобразования отношений базы данных к виду, отвечающему нормальным формам (пошаговый, обратимый процесс приведения данных в более простую и логичную структуру).** **Целью является уменьшение потенциальной противоречивости хранимой в базе данных информации.** **Денормализация базы данных — это процесс обратный от нормализации. Эта техника добавляет избыточные данные в таблицу, учитывая частые запросы к базе данных, которые объединяют данные из разных таблиц в одну таблицу. Необходимо для повышения производительности и скорости извлечения данных, за счет увеличения избыточности данных.** **Каждая нормальная форма включает в себя предыдущую. Типы форм:** **- Первая нормальная форма (1NF) - значения всех полей атомарны (неделимы), нет множества значений в одном поле.** **- Вторая нормальная форма (2NF) - все неключевые поля зависят только от ключа целиком, а не от какой-то его части.** **- Третья нормальная форма (3NF) - все неключевые поля не зависят друг от друга.** **- Нормальная форма Бойса-Кодда, усиленная 3 нормальная форма (BCNF) - когда каждая её нетривиальная и неприводимая слева функциональная зависимость имеет в качестве своего детерминанта некоторый потенциальный ключ.** **- Четвёртая нормальная форма (4NF) - не содержатся независимые группы полей, между которыми существует отношение «многие-ко-многим».** **- Пятая нормальная форма (5NF) - каждая нетривиальная зависимость соединения в ней определяется потенциальным ключом (ключами) этого отношения.** **- Доменно-ключевая нормальная форма (DKNF) - каждое наложенное на нее ограничение является логическим следствием ограничений доменов и ограничений ключей, наложенных на данное отношение.** **- Шестая нормальная форма (6NF) - удовлетворяет всем нетривиальным зависимостям соединения, то есть не может быть подвергнута дальнейшей декомпозиции без потерь. Введена как обобщение пятой нормальной формы для хронологической базы данных.**
28.Что такое TIMESTAMP?	**DATETIME** предназначен для хранения целого числа: YYYYMMDDHHMMSS. И это время не зависит от временной зоны настроенной на сервере. Размер: 8 байт **TIMESTAMP** хранит значение равное количеству секунд, прошедших с полуночи 1 января 1970 года по усреднённому времени Гринвича. Тогда была создана Unix. При получении из базы отображается с учётом часового пояса. Размер: 4 байта
29. Шардирование БД	При большом количестве данных запросы начинают долго выполняться, и сервер начинает не справляться с нагрузкой. Одно из решений, что с этими данными делать — это масштабирование базы данных. Например, шардинг или репликация. **Шардинг бывает вертикальным(партицирование) и горизонтальным.** У нас есть большая таблица, например, с пользователями. Партицирование — это когда мы одну большую таблицу разделяем на много маленьких по какому-либо принципу. Единственное отличие горизонтального масштабирования от вертикального в том, что горизонтальное будет разносить данные по разным инстансам в других базах.
	01. CREATE TABLE news (02. id bigint not null, 03. category_id int not null, 04. author character varying not null, 05. rate int not null, 06. title character varying 07.)
	Есть таблица news, в которой есть идентификатор, есть категория, в которой эта новость расположена, есть автор новости...

	Нужно сделать 2 действия над табличкой — это поставить у нашего шарда, например, news_1, то, что она будет наследоваться от news. Наследованная таблица будет иметь все колонки родителя, а также она может иметь свои колонки, которые мы дополнительно туда добавим. Там не будет ограничений, индексов и триггеров от родителя — это важно. **2-ое действие** — это поставить ограничения. Это будет проверка, что в эту таблицу будут попадать данные только с нужным признаком.
	01. CREATE TABLE news_1 (02. CHECK (category_id = 1) 03.) INHERITS (news)
	Т.е. только записи с category_id=1 будут попадать в эту таблицу. На базовую таблицу надо добавить правило. Когда мы будем работать с таблицей news, вставка на запись с category_id = 1 должна попасть именно в партицию news_1. Правило называем как хотим.
	01. CREATE RULE news_insert_to_1 AS ON INSERT TO news 02. WHERE (category_id = 1) 03. DO INSTEAD INSERT INTO news_1 VALUES (NEW.*)
30.EXPLAIN	Когда вы выполняете какой-нибудь запрос, оптимизатор запросов MySQL пытается придумать оптимальный план выполнения этого запроса. Можно посмотреть этот план используя запрос с ключевым словом EXPLAIN перед оператором SELECT. **EXPLAIN SELECT * FROM categories** После EXPLAIN в запросе вы можете использовать ключевое слово EXTENDED и MySQL покажет вам дополнительную информацию о том, как выполняется запрос. Чтобы увидеть эту информацию, вам нужно сразу после запроса с EXTENDED выполнить запрос SHOW WARNINGS. EXPLAIN **EXTENDED** SELECT City.Name FROM City Затем **SHOW WARNINGS**
31. Как сделать запрос из двух баз?	Если в запросе таблица указывается с именем базы данных database1.table1, то таблица выбирается из database1, если просто table1, то - из активной базы данных. Надо, чтобы базы были на одном сервере. SELECT t1.*, t2.* FROM database1.table1 AS t1 INNER JOIN database2.table2 AS t2 ON t1.field1 = t2.field1
32. Что быстрее убирает дубликаты distinct или group by?	Если нужны уникальные значения - DISTINCT. Если нужно группировать значения - GROUP BY. Если задача заключается именно в поиске дубликатов - **GROUP BY будет лучше.**
33. Механизмы оптимизации запросов в БД	Например, добавить индекс по нужной колонке
34. Что такое «триггер»?	Триггер (trigger) — это хранимая процедура особого типа, исполнение которой обусловлено действием по модификации данных: добавлением, удалением или изменением данных в заданной таблице реляционной базы данных. Триггер запускается сервером автоматически и все производимые им модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Момент запуска триггера определяется с помощью ключевых слов BEFORE (триггер запускается до выполнения связанного с ним события) или AFTER (после события).