**Основные вопросы по SQL на собеседовании**

**Что такое соединения в SQL?**

Соединение — это операция, которая используется для объединения данных из нескольких таблиц в новую таблицу. Различные типы объединений определяют, как данные между таблицами сопоставляются с новой таблицей. Когда вам нужно получить данные из нескольких таблиц в одном запросе, есть большая вероятность, что вы будете использовать операцию соединения.

Интервьюеры задают этот вопрос, чтобы проверить ваше фундаментальное понимание того, как запрашиваются данные и насколько это необходимо практически для любого типа приложений. Мы не можем просто свалить все данные в одну таблицу, так как она станет громоздкой и плохо организованной.

Пример: мы хотели бы хранить таблицу клиентов (с их именем, адресом, компанией, которую они представляют и т. д.) отдельно от таблицы транзакций (с купленными товарами, когда была совершена транзакция, сколько стоили товары, кто совершил покупка и др.).

Когда мы сохраняем информацию о том, кто совершил покупку, мы не хотели бы дублировать всю информацию о покупателе в каждой строке таблицы транзакций, и мы хотели бы сохранить только идентификатор клиента. Чтобы ответить на такой вопрос, как «сколько покупок было совершено клиентами, проживающими в определенном почтовом индексе?», нам нужно соединить две таблицы, чтобы получить такой ответ:

**SELECT** **COUNT**(\*) **FROM** customers **c** **JOIN** transactions **t** **ON** **t**.customer\_id = **c**.**id** **WHERE** **c**.zipcode = 94107;

**В чем разница между DELETE и TRUNCATE заявления?**

Это один из тех вопросов на собеседовании по SQL, которые призваны оценить ваше понимание того, как строки базы данных хранятся и управляются внутри. Хотя DELETE и TRUNCATE оба могут использоваться для удаления всех данных из таблицы, база данных обрабатывает эти запросы по-разному. DELETE операции могут фильтровать целевые строки, поскольку они поддерживают предложение WHERE , тогда как TRUNCATE операция удаляет всю таблицу.

При удалении всей таблицы с помощью этих двух операций TRUNCATE выполняется быстрее за счет невозможности выполнения некоторых операций, зависящих от журнала транзакций. Например, некоторые SQL-серверы могут откатывать DELETE операции, а не TRUNCATE операции.

Как правило, вы должны использовать эту DELETE операцию из-за таких преимуществ, как история в журнале транзакций. В ситуациях, когда DELETE операция над всей таблицей занимает слишком много времени, вы можете рассмотреть возможность использования этой TRUNCATE операции. Интервьюеру было бы интересно услышать о компромиссах между двумя операциями.

**В чем разница между первичным ключом и уникальным ключом?**

Первичные ключи должны быть уникальными и использоваться для идентификации записей таблицы, тогда как уникальные ключи служат ограничениями в данных таблицы. Понимание уникальных ключей показывает интервьюеру, что вы понимаете некоторые важные способы поддержания целостности данных таблицы базы данных.

Например, когда вы разрабатываете таблицу для отслеживания пользователей, у вас может быть столбец для user\_id в качестве первичного ключа. user\_id будет уникальным и используется для ссылки на остальные данные в строке. Чтобы поддерживать качество данных, у вас может быть столбец для в phone\_number качестве уникального ключа, чтобы гарантировать, что в таблице может существовать только один номер каждого телефона.

**Какова цель внешнего ключа в SQL?**

Внешние ключи — это столбцы, используемые для ссылки на первичные ключи в других таблицах. Это помогает установить ограничение для столбца, чтобы обеспечить ссылочную целостность таблиц. Внешние ключи являются фундаментальной концепцией проектирования таблиц базы данных, и интервьюеры захотят убедиться, что вы их понимаете и использовали в прошлом. Понимание их показывает, что вы знаете, как проектировать схемы, охватывающие несколько таблиц базы данных.

Например, представьте, что у вас есть две таблицы: заказы и пользователи. Каждый заказ должен быть создан пользователем, чтобы таблица заказов могла иметь внешний ключ для первичного ключа пользовательской таблицы. Это ограничение гарантирует, что каждый пользователь, определенный в таблице заказов, ссылается на допустимую строку.

**Какие существуют способы оптимизации запроса?**

Планы запросов — это полезный способ изучить, как будет выполняться запрос, и помочь вам лучше понять, что замедляет выполнение вашего запроса.

Например, добавление к запросу в PostgreSQL символа EXPLAIN покажет план запроса для команды. Это покажет сканирование таблицы, которое будет задействовано в запросе.

Итак, если мы хотим проверить поведение запроса: SELECT \* FROM table\_1;, we can run the query EXPLAIN SELECT \* FROM table\_1;. Оттуда план запроса разбивает различные шаги, которые будут предприняты для выполнения запроса. Это распространенный способ выявления ненужных полных сканирований таблицы, который можно облегчить, настроив правильные индексы.

Эти методы важны, потому что интервьюеры хотят понять, как вы можете подходить к устранению неполадок запросов к базе данных. Если кандидат знает только, как выполнить запрос к базе данных для измерения производительности, это свидетельствует о недостатке знаний и опыта работы с реляционными базами данных.

**Какими способами можно определить, как можно оптимизировать запрос?**

Запросы можно оптимизировать разными способами. Вот несколько распространенных примеров:

* Уменьшите объем данных для запроса с помощью WHERE предложений.
* Ограничьте количество полезных строк, которые база данных должна запрашивать с помощью LIMIT предложения.
* Добавьте индекс для часто запрашиваемых столбцов.

Цель этого вопроса часто состоит не в том, чтобы просто перечислить все приведенные выше примеры. Поскольку этот вопрос является открытым, некоторые интервьюеры могут представить этот вопрос со сценарием, в котором можно определить и применить определенные оптимизации.

**Что такое нормализация и в чем ее преимущества?**

Нормализация базы данных — это стратегия эффективной организации данных в базе данных. Цель состоит в том, чтобы уменьшить количество избыточных данных, чтобы одни и те же данные не хранились в нескольких таблицах. Вместо этого на данные будет ссылаться первичный ключ.

Например, таблица с именем *orders* может иметь столбец с именем *user\_id* . Вместо того, чтобы хранить копию пользовательских данных в таблице заказов, мы можем просто ссылаться на эти данные из другой таблицы с помощью соединения.

Его часто просят оценить понимание кандидатом дизайна таблицы. Нормализация данных является ключевым компонентом проектирования схем таблиц в реляционных базах данных.

**Что такое сущности и отношения?**

Объект представляет собой абстракцию набора связанных данных и представлен в виде таблиц *. Отношения* определяют, как объекты связаны друг с другом.

Например, предположим, что у нас есть две таблицы с именами orders и users. Наши orders и users являются нашими сущностями. Мы можем представить, что у одного пользователя может быть много заказов. Таким образом, пользователи могут иметь отношения «один ко многим» с заказами.

Сущности и отношения часто используются в процессе разработки схемы таблицы. Понимая, как определять объекты и отображать их отношения, вы показываете, что можете продуктивно работать в коллективном сеансе проектирования таблиц базы данных.

**Как можно настроить таблицу так, чтобы запросы к определенным строкам не приводили к полному сканированию таблицы?**

Полного сканирования таблицы можно избежать, запрашивая индексированные столбцы и используя ограничения. Индексированные столбцы помогают базе данных оптимизировать поиск в таблицах.

Существует множество стратегий сокращения запрашиваемых строк, и их можно подтвердить с помощью планов запросов. Интервьюеры часто задают этот вопрос о SQL, чтобы оценить ваше понимание того, как правильно проектировать таблицы SQL и оптимизировать запросы.

**Дополнительные вопросы и ответы на собеседовании по SQL**

**Объясните некоторые различные типы индексов в SQL.**

В кластеризованном индексе данные физически хранятся в одной и той же области таблицы, поэтому их можно эффективно запрашивать и извлекать вместе. Некластеризованный индекс часто используется для запросов на основе ключей, тогда как кластеризованный индекс часто используется для диапазонов.

Большинство баз данных SQL выберут лучший тип индекса для вашего варианта использования. Интервьюеры обычно задают этот вопрос, чтобы оценить ваше понимание внутренней настройки индексов.

**Каков сценарий, когда вы решите использовать ноль или пробел над NULL значением в строке?**

Использование нуля или пробела над NULL значением является конструктивным решением. NULL может означать отсутствие данных.

Пример можно привести с отчеством: если пользователь не указал отчество, то его отчество будет NULL. Если пользователь указал, что у него нет отчества, то его отчество будет пустой строкой.

Если мы знаем, что никогда не будем заботиться о различии между значением нуля/пробела и отсутствием значения, тогда мы можем пойти дальше и по умолчанию установить для столбца нулевое значение или значение пробела. Это может быть полезно в таблице, которая отслеживает, сколько раз что-то произошло (например, количество посещений веб-сайта). Нам не нужно различать 0 и отсутствие данных, поскольку отсутствие данных подразумевает 0, поэтому мы можем упростить код нашего приложения, установив по умолчанию 0.

Технический рекрутер или менеджер по найму может задать подобный вопрос на собеседовании по SQL, чтобы оценить ваше понимание того, как различные типы хранятся в таблице базы данных. Возможность сообщить об этих решениях показывает, что вы понимаете некоторые соображения по выбору подходящих компромиссов в дизайне.

**В чем разница между перекрестным соединением и естественным соединением?**

Если не указано, соединение по умолчанию будет использовать внутреннее соединение. Хотя есть вероятность, что вы раньше не использовали перекрестное соединение или естественное соединение, понимание различных типов соединений может показать интервьюеру, что вы знакомы с различными способами объединения данных между таблицами.

Перекрестное соединение будет производить декартово произведение между двумя таблицами. Обычно он используется, когда вы хотите создать комбинацию, используя каждую строку из двух таблиц. Например, если вы хотите найти все комбинации цветов между различными предметами одежды, вы должны использовать перекрестное соединение для получения каждой комбинации.

Естественное соединение создаст таблицу, которая объединяет столбцы с одинаковыми именами и типами. Любой общий столбец между таблицами будет рассматриваться для объединения. Это отличается от обычно используемого внутреннего соединения, где соединения явно выполняются для указанных столбцов. Важно отметить, что если естественное соединение не находит совпадающих столбцов, оно по существу дает тот же результат, что и перекрестное соединение.

**Что такое UNION, MINUSи INTERSECT команды?**

Эти три команды известны как операции над множествами. Интервьюеры зададут вам этот вопрос, чтобы оценить ваше знакомство с анализом данных и обработкой данных с использованием SQL. Если вы широко использовали SQL, но не сталкивались с этими командами, интервьюер может узнать, что вы можете использовать команды SQL для выполнения операций CRUD вместо анализа данных.

* UNION создаст таблицу, в которой данные содержатся в двух таблицах.
* MINUS создаст таблицу, в которой есть данные из первой таблицы за вычетом данных из второй таблицы.
* INTERSECT создаст таблицу с общими данными между двумя таблицами.

Если мы представим диаграмму Венна с двумя пересекающимися окружностями и тремя отдельными частями, UNION представляющими все три части, MINUS представляющими левую часть и INTERSECT представляющую среднюю часть.

**Напишите SQL-запрос, чтобы получить третью по величине зарплату сотрудника из employee\_table.**

SELECT salary FROM employee\_table ORDER BY salary DESC LIMIT 1 OFFSET 2;

* SELECT salary указывает, что мы хотим вернуть только зарплату и не включать другие столбцы.
* ORDER BY salary DESC возвращает все результаты, упорядоченные от наибольшей до наименьшей зарплаты.
* LIMIT 1 указывает, что мы хотим получить только одну строку. Поскольку результаты уже упорядочены фильтром ORDER BY , это вернет наибольшую зарплату.
* OFFSET 2 указывает, что мы хотим пропустить первые 2 строки.

Этот вопрос состоит из общих команд в SQL. Вы должны быть знакомы со всеми этими командами, чтобы эффективно писать SQL-запросы.

Обратите внимание, что работодатель может задавать вам похожие вопросы на собеседовании по SQL с разной формулировкой. Цель состоит не в том, чтобы запомнить это решение и повторить его интервьюеру. Когда вам задают этот тип вопроса, вы должны разбить свой запрос на части, которые приводят к окончательному решению. Это покажет вашему интервьюеру, что у вас дисциплинированный процесс написания запросов.

**Зачем нужны групповые функции в SQL?**

Групповые функции — один из ключевых способов выполнения анализа данных с помощью SQL. Интервьюер задаст этот вопрос, чтобы определить, используете ли вы SQL для CRUD или использовали ли вы SQL для анализа данных.

Групповые функции помогают нам объединить набор строк в одну группу данных, представленных строками. Эти функции часто используются для анализа таблиц, чтобы лучше понять данные, которые они представляют.

Например, предположим, что у нас есть таблица users , в которой есть строка с именем country , указывающая страну, из которой находится пользователь. Используя GROUP BY, мы можем определить количество пользователей для каждой страны в нашей таблице.

Отвечая на этот вопрос интервью для SQL, также важно предоставить конкретные примеры того, как вы, возможно, использовали групповые функции раньше в прошлом.

**Для чего используются функции SQL?**

Функции SQL предоставляют способы выполнения вычислений в базе данных. Они могут включать агрегации, которые часто используются для аналитики. Примером функции агрегирования может быть AVG функция, которая возвращает средние цены, уплаченные за покупку: SELECT AVG(price) AS average\_price FROM purchases.

Когда вам зададут этот вопрос, интервьюер будет впечатлен, если вы сможете привести несколько примеров функций SQL, которые вы использовали, и как они использовались в вашем запросе.

**В чем разница между HAVING оговоркой и WHERE оговоркой?**

WHERE используется для фильтрации строк перед их возможной группировкой. HAVING используется для фильтрации строк после их группировки. HAVING функции очень похожи на , WHERE но используются после некоторой формы агрегации. WHERE Предложения должны быть предпочтительными, когда это возможно, так как наш запрос будет быстрее, если мы отфильтруем предварительную агрегацию данных, чем последующую, но некоторые фильтры могут выполняться только после агрегации и поэтому требуют использования предложения WHERE .

**Пример** . Учитывая таблицу заказов, верните число customer id и количество заказов, сделанных среди клиентов, совершивших не менее 10 покупок. Единственный способ узнать, сколько заказов сделал клиент, — это сначала подсчитать (объединить) все его заказы перед фильтрацией. Мы могли бы написать этот запрос как SELECT customer\_id FROM (SELECT COUNT(\*) AS count, customer\_id from orders GROUP BY 2) WHERE COUNT >= 10, или мы можем упростить его с помощью HAVING предложения: SELECT COUNT(\*) AS count, customer\_id from orders GROUP BY 2 HAVING count >= 10

Интервьюер может не спрашивать вас явно о разнице между этими двумя пунктами, но представить вопрос в более широком контексте. Они могут попросить вас написать запрос, в котором для возврата правильного результата необходимы оба предложения WHERE и .HAVING

**Как вы можете получить альтернативные записи из таблицы?**

Есть несколько способов получить чередующиеся записи. Цель интервьюера — оценить ваше знакомство и удобство при написании SQL-запросов и использовании функций.

Обычно таблицы имеют автоматически увеличивающиеся первичные ключи. Мы можем использовать оператор модуля % для получения чередующихся строк. Это работает, потому что число, деленное на 2, всегда возвращает либо 0, либо 1.

SELECT \* FROM <table\_name> WHERE <table\_id> % 2 = 0;

SELECT \* FROM <table\_name> WHERE <table\_id> % 2 = 1;

**Назовите оператор, который используется в запросе на сопоставление с образцом.**

LIKE используется для сопоставления строк. Различные формы LIKE операций могут быть доступны в разных реализациях базы данных SQL. Например, LIKE часто может использоваться для сопоставления с образцом без учета регистра.

Если у нас есть таблица products с именем столбца с именем name, мы можем искать в каждой строке продукты с именами, содержащими слово *toy* : SELECT \* FROM products WHERE name LIKE '%toy%.

Сопоставление с образцом часто используется для поиска определенного текста в столбце по многим строкам. Интервьюеры будут рады узнать, что вы понимаете, как работает сопоставление с образцом и как оно часто применяется в производственных базах данных.

**Когда было бы более уместно использовать материализованное представление вместо представления?**

Как материализованные представления, так и представления упрощают запрос данных, преобразуя некоторые табличные данные в собственную структуру. Материализованные представления сохраняют данные, тогда как представления вычисляют свои данные при каждом запросе.

Это означает, что материализованные представления имеют лучшую производительность чтения, поскольку данные сохраняются. Когда производительности достаточно, иногда рекомендуется использовать представление, чтобы уменьшить объем данных, которые будут храниться с материализованным представлением.

Когда менеджеры по найму и технические рекрутеры задают подобные вопросы на собеседовании по SQL, они заинтересованы в том, чтобы знать, что вы понимаете, как добиться компромиссов в производительности в SQL. Даже если интервьюер не задает именно этот вопрос, вы можете сослаться на преимущества материализованного представления или представления в ситуации, когда вам нужно создавать сложные запросы.

**Перечислите некоторые преимущества и недостатки хранимой процедуры.**

Хранимые процедуры полезны для многократно используемых SQL-запросов. На практике мы можем создать хранимую процедуру вместо хранения списка многократно используемых SQL-запросов. Это уменьшает возможность ошибки и стандартизирует набор операций для воспроизводимости.

Однако хранимые процедуры часто очень специфичны для случая использования и не переносимы. Хранимая процедура не будет работать в другой таблице базы данных с аналогичной структурой, и ее необходимо будет создать заново, чтобы она заработала.

Поскольку они являются пользовательскими, их сложно тестировать и интегрировать с другими инструментами. Разработчики часто используют инструменты, расположенные поверх баз данных, чтобы упростить разработку приложений — известно, что хранимые процедуры несовместимы с некоторыми из этих инструментов.

Чтобы сделать ваш ответ на этот вопрос интервью SQL более эффективным, сошлитесь на пример, когда вы, возможно, использовали или рассматривали возможность использования хранимой процедуры. Интервьюеры, как правило, больше впечатляются, когда ответ кандидата подкрепляется соответствующим опытом, а не определениями.

**В чем разница между базами данных OLTP (например, MySQL) и OLAP (например, хранилище данных)?**

Это еще один из тех вопросов для собеседования с разработчиками SQL, которые призваны помочь интервьюеру оценить, понимает ли кандидат различные базы данных SQL, которые могут существовать, и чем они отличаются друг от друга. Различные варианты использования часто определяют тип используемой базы данных SQL.

Базы данных OLTP предназначены для быстрых запросов с надежной целостностью данных. Как правило, они оптимизированы для выполнения повседневных бизнес-операций, связанных с чтением и записью в реальном времени.

Базы данных OLAP оптимизированы для автономной аналитики и обычно включают сложные агрегаты. По сравнению с базами данных OLTP они часто имеют меньшие объемы запросов, содержат больше исторических данных и требуют значительно меньшего количества правок.

Если мы проиллюстрируем это на веб-приложении, база данных OLTP, вероятно, обрабатывает метаданные для веб-приложения. Действия пользователя и данные сохраняются и извлекаются для базы данных OLTP. Базы данных OLAP, вероятно, будут хранить данные того же типа, но будут использоваться для понимания ключевых показателей, таких как удержание пользователей и их поведение.

Ваш ответ на этот вопрос SQL будет гораздо более эффективным, если вы предоставите примеры различных баз данных OLTP и баз данных OLAP, которые вы, возможно, использовали.