Тест SQL (Удалённое тестирование)

1. **Что такое транзакция?**

Транзакция – набор операций, которые выполняются как единое целое. Принцип таков, что все операции должны быть выполнены успешно, или ни одна из них не должна быть выполнена. Транзакция представляет собой логическую единицу работы с базой данных, которая позволяет поддерживать целостность данных и обеспечивает атомарность, согласованность, изолированность и надежность (ACID-свойства) операций. ACID – это набор свойств, которые гарантируют корректность выполнения транзакций:

* Атомарность (Atomicity) – гарантирует, что транзакция не может выполниться частично. База данных не допускает промежуточных состояний. Все операции транзакции либо выполняются все вместе, либо ни одна из них не выполняется. Если внутри транзакции с какой-то одной операцией произошел сбой, то база откатывает всю транзакцию назад и приводит систему к исходному состоянию. Пример: если деньги с карты списались, то они кому-то поступили, либо деньги вернулись обратно на карту. Если мы покупаем билет в кинотеатр, и в процессе покупки произошел какой-то сбой в системе, то билет не должен быть куплен.
* Согласованность (Consistency) – транзакция должна приводить базу данных из одного согласованного состояния в другое согласованное состояние. Достигается через ограничения (CONSTRRAINTS). Пример: если клиент удаляется из базы, то его заказы не должны повиснуть в воздухе, не привязанные к покупателю. Поэтому важно проставлять внешний ключ. Если мы не повесили внешний ключ, то в будущем целостность данных может быть нарушена;
* Изолированность (Isolation) – параллельное выполнение нескольких транзакций не должно негативно влиять на их результаты. При параллельном выполнении разные транзакции не должны влиять друг на друга. Проблем нет, если операции транзакции взаимодействуют с непересекающимися данными. Но если в один момент времени две и более транзакции работают с одним и тем же набором данных, то возникнет явление race condition (состояние гонки). Для состояния гонки существует 4 эффекта:

1. Эффект потерянного обновления. Когда две и более транзакций выполняются параллельно, и при фиксировании результатов одной транзакции, результат другой транзакции теряется.

Решается через блокировку Read uncommitted на первом уровне изоляции. Все успешно выполненные транзакции сформируют итоговое значение в записи. Команды изменения одних и тех же строк, запущенные параллельно, фактически выполнятся последовательно, и ни одно из изменений не потеряется. Транзакции, выполняющие только чтение, при данном уровне изоляции никогда не блокируются.

1. Грязное чтение. Происходит, когда изменения первой транзакции не были зафиксированы, а вторая транзакция стала работать с незафиксированными изменениями. Соответственно, если изменения первой транзакции откатятся назад (ROLLBACK), то вторая транзакция продолжит оперировать неактуальными данными.

Решается через блокировку Read committed на втором уровне изоляции. Транзакция, которая изменяет данные, блокирует данные для считывающих транзакций, до своего завершения, препятствуя, таким образом, «грязному» чтению.

1. Эффект неповторяемого чтения. Возникает, когда одна транзакция считывает данные, затем происходит вторая транзакция (выполнение UPDATE или DELETE) с фиксированием результата, и первая транзакция считывает новый результат. Получает первая транзакция два раза считала одну и ту же строку и вернула разные результаты.

Решается через блокировку Repeatable read на третьем уровне изоляции. Читающая транзакция «не видит» изменения данных, которые были ею ранее прочитаны. При этом никакая другая транзакция не может изменять данные, читаемые текущей транзакцией, пока та не окончена. Не действует на INSERT, поэтому сохраняется риск чтения фантомов.

1. Эффект чтения фантомов. Суть та же: как и в случае с неповторямым чтением, одна транзакция выдает два разных результата из-за коммита второй транзакции. Отличие в том тем, что результат повторного обращения к данным изменился не из-за изменения/удаления самих этих данных, а из-за появления новых (INSERT), то есть «фантомных» данных.

Решается через блокировку Serializable (упорядочиваемость)на четвертом уровне изоляции. Чтение фантомов исключается на самом высоком уровень изолированности транзакций. Результат выполнения нескольких параллельных транзакций должен быть таким, как если бы они выполнялись последовательно.

* Долговечность (Durability) – результаты успешно завершенной транзакции должны сохраняться в базе данных даже в случае отказа системы. Если пользователь получил уведомление об успешной операции, то изменения не отменяются из-за какого-то сбоя. Но если сбой произошел до уведомления об успешной операции, то транзакция не выполнится (атомарность).

SQL-транзакции обычно начинаются с команды BEGIN TRANSACTION и завершаются с командой COMMIT или ROLLBACK, в зависимости от результата выполнения операций.

1. **Какая разница между CHAR, VARCHAR**

CHAR и VARCHAR - это два типа данных, используемых для хранения символьных строк.  
CHAR - это строка фиксированной длины, которая может содержать максимум 255 символов. Если заданная строка короче, то оставшееся место заполняется пробелами. Если заданная строка длиннее, то она будет обрезана до заданной длины. Например, если создать поле типа CHAR(10), то оно будет занимать 10 байт, независимо от того, сколько символов будет введено.

VARCHAR - это строка переменной длины. Если заданная строка короче, то оставшееся место не заполняется. Если заданная строка длиннее, то она будет сохранена полностью. Например, если вы создадите поле типа VARCHAR(10), то оно будет занимать столько байт, сколько символов будет введено.

1. **Какая разница между UNION и UNION ALL**

UNION объединяет результаты двух или более запросов, и удаляет дубликаты строк.

UNION ALL тоже объединяет результаты двух и более запросов, но не удаляет дубликаты строк. Поэтому команда имеет в своем названии слово ALL, что означает «всё, в том числе и повторяющиеся строки». UNION работает медленнее UNION ALL, так как ему нужно выполнить дополнительные операции на удаление дубликатов.

1. **Что такое Primary Key / Foreign key?**

* PK – идентификатор, который накладывает ограничение уникальности на записи в таблице. Primary key однозначно определяет каждую запись в таблице. Он имеет следующие характеристики:

1. Уникальность: Каждое значение первичного ключа должно быть уникально. Это означает, что никакие две записи не могут иметь одинаковые значения в поле с PK.

2. Ненулевое значение: Значение PK не может быть NULL, то есть каждая запись в таблице должна иметь непустое значение.

3. Неизменяемость: Значение PK не должно изменяться после его создания. Это гарантирует постоянную идентификацию записи в таблице.

* FK – правило, которое использует СУБД чтобы обеспечить контроль целостности и непротиворечивости данных. Работа внешнего ключа в SQL заключается в создании связи между двумя таблицами: ссылочной таблицей (таблица, содержащая первичный ключ) и ссылающейся таблицей (таблица, содержащая внешний ключ). Механизм FK проверяет что значение в поле с внешним ключом существует в списке значений другой таблицы или является NULL. Если данные не соответствуют этим правилам, то при изменении или удалении данных, СУБД выдаст ошибку.

При добавлении или изменении значения в поле с внешним ключом, система проверяет, существует ли новое значение в поле с PK в ссылочной таблице. Если новое значение не найдено в ссылочной таблице, то операция добавления или изменения будет отклонена или вызовет ошибку.

При удалении значения из ссылочной таблицы, система проверяет, отсутствуют ли ссылки на это значение в ссылающейся таблице. Если в ссылающейся таблице найдены ссылки на удаляемое значение, то операция удаления будет отклонена или вызовет ошибку, чтобы сохранить целостность данных. Таким образом, внешний ключ в SQL обеспечивает связанность данных между двумя таблицами и контролирует значения в полях.

Правило внешнего ключа можно дополнить опциями каскадной операции ON DELETE и ON UPDATE, которые позволяют автоматически выполнять определенные действия при удалении или обновлении значения в ссылочной таблице. Есть 4 возможных действия при удалении или изменении данных:

1. CASCADE: Удаление/обновление значения в таблице с PK приведет к автоматическому удалению/обновлению всех связанных через внешний ключ значений в ссылающейся таблице.

2. SET NULL: Удаление/обновление значение в таблице с PK приведет к установке связанных значений в столбцах с внешним ключом в ссылающейся таблице в значение NULL.

3. SET DEFAULT: Удаление/обновление значения в таблице с PK приведет к установке связанных значений в столбцах с внешним ключом в ссылающейся таблице на значение по умолчанию, определенное при создании таблицы.

4. NO ACTION: Удаление/обновление значения в таблице с PK будет отклонено, если существуют связанные значения в ссылающейся таблице.

Например, есть таблица заказов и таблица клиентов. В таблице заказов есть внешний ключ, связывающий ее с таблицей клиентов. При изменении id клиента в таблице клиентов, опция ON UPDATE CASCADE обновит также id клиента во всех связанных записях в таблице заказов. Таким образом, можно поддерживать целостность данных в вашей базе данных.

1. **Чем таблица (Table) отличается от представления (View)?**

* Таблица – это объект базы данных, который состоит из полей которые определяют тип данных, и записей, которые содержат фактические значения для каждого поля.
* Представление – это виртуальная таблица, содержимое которой берется из других таблиц.

1. Обычные (базовые) таблицы содержат реальные данные.

Представление не содержат данных, оно извлекает данные тогда когда мы обращаемся к представлению. Как и у таблицы, у каждого представления, есть свое имя.

2. Таблица создается командой CREATE TABLE [IF NOT EXISTS] table\_name.

Представление создается другой командой: CREATE [OR REPLACE] VIEW view\_name.  
«OR REPLACE» означает, что если уже существует представление с таким названием, то новое представление заменит старое.

3. Наполнение таблицы осуществляется командой INSERT INTO table\_name.

Наполнение представления задается с помощью оператора AS SELECT.

4. Ограничения в представлениях отличаются от ограничений таблиц (PK, PF, UNIQUE, NOT NULL, CHECK, DEFAULT)

У представлений есть собственные ограничения:

1. WITH READ ONLY предотвращает любые операции изменения данных (вставка, обновление, удаление) через представление.

2. WITH CHECK OPTION предотвращает вставку, обновление или удаление данных через представление, если эти данные не соответствуют определенному условию.

3. WITH LOCAL CHECK OPTION предотвращает вставку, обновление или удаление данных, если эти данные не соответствуют условиям. Например, если представление выбирает только записи со значением «Активный» в поле "Статус" и стоит опция WITH LOCAL CHECK OPTION, то будет невозможно вставить, обновить или удалить запись со статусом 'Неактивный'.

Ограничение WITH LOCAL CHECK OPTION применяется только к подчиненным (child) представлениям, но не к родительским (parent) представлениям. Это означает, что любые операции изменения данных через подчиненное представление, которые нарушают условие, будут заблокированы. Однако операции изменения данных через родительское представление не будут проверяться.

4. WITH CASCADED CHECK OPTION предотвращает вставку, обновление или удаление данных, если эти данные не соответствуют условиям, прописанным как в родительском, так и в подчиненном представлениях.

Например, есть главное представление parent\_view которое выбирает только записи с зарплатой >100000 руб. Есть подчиненное представление child\_view которое отфильтровывает результат родительского представления по зарплатам отделу продаж. Любые операции изменения данных, которые нарушают условия сразу в обоих представлениях (и по полю salary в parent\_view и по полю department в child\_view) будут невозможны.

Если данные в базовой таблице изменились, то при обращении к представлению мы получим новые данные.

Кэширование результатов не производится, мы забираем данные в момент обращения к представлению.

С помощью представлений можно разграничить доступ к данным. Закрыть пользователям доступ к базовой таблице, но предоставить к некой выборке. Результат выборки упаковать в представление.

Специфика ограничений зависят от конкретной СУБД либо версии СУБД.

Например в MySQL <https://dev.mysql.com/doc/refman/8.0/en/create-view.html> и PostgreSQL <https://www.postgresql.org/docs/current/sql-createview.html> нет WITH READ ONLY. Доступ только к чтению задается через разграничение прав.

Также, на обычное представление нельзя повесить триггер. Триггер можно повесить только на материализованное представление (MATERIALIZED VIEW).

Есть два вида представлений. Обычное и материализованное. В MySQL – только обычное.

Обычное представление результат не сохраняет (выборка осуществляется в момент обращения).

Материализованное представление, наоборот, сохраняет результат для дальнейшего использования.

Поскольку материализованное занимает сохраняет результат, то оно занимает место в памяти.

Обычное преставление проигрывает в производительности запроса. Материализованное представление выигрывает в скорости, но требует планировщик-шедулер или триггер чтобы актуализировать данные.

Представление нужны для выдачи определенных данных пользователям, и для ускоренного обращения к часто используемым запросам.

Представления (как и индексы) обычно не используются если база данных маленькая.

1. **Есть две таблицы «Офисы» (Office) и «Города» (City)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Office** | | |  | **City** | |
| **Office\_Id** | **Name** | **City\_id** |  | **City\_id** | **Name** |
| 1 | Office\_M1 | 1 |  | 1 | Moscow |
| 2 | Office\_L1 | 2 |  | 2 | London |
| 3 | Office\_M2 | 1 |  | 3 | Paris |
| 4 | Office\_R1 | 4 |  |  |  |
| 5 | Office\_R2 | 4 |  |  |  |

**Сколько строк вернут запросы:**

|  |  |
| --- | --- |
| SELECT \* FROM Office o LEFT JOIN City c ON o.City\_id=c.City\_id | 5. Обычная логика работы LEFT JOIN |
| SELECT \* FROM Office o INNER JOIN City c ON o.City\_id=c.City\_id | 3. Обычная логика работы INNER JOIN |
| SELECT \* FROM Office o OUTER JOIN City c ON o.City\_id=c.City\_id | 0. Нужно уточнение LEFT или RIGHT, просто OUTER вернет ошибку |
| SELECT \* FROM Office o CROSS JOIN City c ON o.City\_id=c.City\_id | 0. CROSS JOIN не требует указания ключей. |
| SELECT \* FROM Office o FULL JOIN City c ON o.City\_id=c.City\_id | 6. Три записи будут от INNER JOIN. Две записи будут от LEFT, и последняя запись 3 Paris будет от RIGHT |
| SELECT \* FROM Office o RIGHT JOIN City c ON o.City\_id=c.City\_id | 4. Обычная логика работы RIGHT JOIN |
| SELECT \* FROM Office o, City c | 0. Вернет ошибку. Нет ключей. Их надо указать в WHERE. См. Ниже. |
| SELECT \* FROM Office o, City c WHERE o.City\_id=c.City\_id | 3. Аналог INNER JOIN |
| SELECT \* FROM Office o  UNION ALL SELECT \* FROM City c | 0. Вернет ошибку. Кол-во полей для UNIONов должно быть одинаковым. |
| SELECT Office\_Id FROM Office o  UNION  SELECT City\_id FROM City c | 5. Поскольку указано по одному полю, то их кол-во совпадает. |

1. **Есть таблица T**

|  |  |
| --- | --- |
| **Id** | **A** |
| -1 | Mary |
| 2 | 3 |
| 3 | 3 |
| -4 | <NULL> |
| 5 | -2 |
| 6 | <NULL> |

**Какой результат вернут запросы:**

|  |  |
| --- | --- |
| Select count(\*) from T | 6. Посчитает общее кол-во записей |
| Select count(A) from T | 4. Посчитает кол-во записей в поле А, но NULL не учтет |
| Select count(distinct A) from T | 3. Посчитает кол-во уникальных значений в поле А, но NULL не учтет |
| Select count(distinct \*) from T | 0. Distinct требует после себя не астериск, а конкретное поле. |
| Select SUM(1) from T | 6. Функция SUM() находит сумму всех значений в столбце, но в данном случае нет необходимого столбца в запросе. Вместо этого, функция `SUM` просто складывает переданное значение "2" столько раз, сколько строк в таблице. В данном случае, в таблице 6 строк, поэтому результат будет 2+2+2+2+2+2 = 12. |

**Сколько строк вернут запросы:**

|  |  |
| --- | --- |
| SELECT \* FROM T WHERE A = NULL | 0 строк. Правильно писать IS NULL |
| SELECT \* FROM T WHERE id >= A | Зависит от СУБД. MySQL: 2 строки  PostgreSQL: ошибку, потребует преобразовать типы в INT |
| SELECT COUNT(\*), A FROM T GROUP BY A | 4 строки. Потому что в поле A есть 4 уникальных значения |

1. **Получить максимальное значение поля ID в таблице T без использования функции max()**

SELECT Id

FROM T

ORDER BY id DESC

LIMIT 1

1. **Выбрать записи с четными ID**

SELECT Id

FROM T

WHERE id % 2 = 0

1. **Посчитать количество положительных и отрицательных значений столбца ID в таблице T одним запросом**

SELECT

COUNT(CASE WHEN id > 0 THEN id END) positive,

COUNT(CASE WHEN id < 0 THEN id END) negative

FROM T