Тема 7. Транзакции и целостность данных. Понятие транзакции, ACID свойства. Управление транзакциями.

Цель занятия:

Получить представление о том, как использовать транзакции и ограничения для обеспечения целостности данных в MySQL.

Учебные вопросы:

1. Понятие транзакции.
2. ACID-свойства транзакций.
3. Управление транзакциями.
4. Триггеры как механизм обеспечения целостности данных.
5. Заключение.

1. Понятие транзакции.

Транзакция — это единица работы с базой данных, представляющая собой последовательность операций, которые выполняются как одно целое. Транзакция либо выполняется полностью, либо не выполняется вовсе. Транзакции обеспечивают согласованность данных даже в случае сбоев или ошибок.

Ключевые характеристики транзакции (ACID):

- **Атомарность** (Atomicity): Все операции в транзакции должны быть выполнены полностью или не выполнены вообще.
- **Согласованность** (Consistency): Транзакция должна переводить базу данных из одного согласованного состояния в другое.
- **Изолированность** (Isolation): Результаты транзакции не должны быть видимы другим транзакциям до её завершения.
- **Долговечность** (Durability): Изменения, внесённые транзакцией, сохраняются даже в случае сбоя системы.

Примеры использования транзакций в реальных задачах

- **Банковские переводы**: Когда клиент переводит деньги с одного счёта на другой, транзакция включает несколько шагов: снятие денег с одного счёта и зачисление на другой. Если какая-то часть операции не удастся (например, деньги списались, но не были зачислены), система должна откатить транзакцию, чтобы сохранить согласованность данных.
- **Интернет-магазин**: При оформлении заказа транзакция может включать сразу несколько операций: обновление статуса товара на "в резерве", обновление информации о платеже и создание записи о доставке. Если хотя бы один из этих шагов не удался, вся операция должна быть отменена.
- **Обновление данных в нескольких таблицах**: При обновлении данных, которые затрагивают несколько связанных таблиц, транзакции гарантируют, что изменения во всех таблицах будут согласованными. Например, при изменении информации о сотруднике в таблицах "Сотрудники" и "Зарплаты" изменения должны произойти одновременно, чтобы не возникло несоответствий.
- **Резервное копирование базы данных**: Для создания надежной копии данных можно использовать транзакции, чтобы гарантировать, что во время резервного копирования все данные остаются в согласованном состоянии и никакие изменения не произойдут.

Использование транзакций критично в ситуациях, где необходима высокая степень защиты данных от ошибок или сбоев, таких как банковские операции, бронирование билетов, учет товаров и т. д.

2. ACID-свойства транзакций.

ACID — это набор свойств, которые обеспечивают надёжность выполнения транзакций в базах данных. Эти свойства помогают сохранить целостность данных в случае сбоев, ошибок или одновременного доступа к базе данных несколькими пользователями.
Расшифровка и объяснение каждого свойства:

Атомарность (Atomicity).

Операции в транзакции выполняются как одно целое. Либо все изменения, сделанные в рамках транзакции, фиксируются, либо, в случае ошибки, откатываются.

Пример: Если в банковской транзакции происходит перевод денег, система либо успешно списывает деньги с одного счёта и зачисляет их на другой, либо, при ошибке, все действия отменяются.

Согласованность (Consistency)

Транзакция переводит базу данных из одного согласованного состояния в другое, не нарушая правил целостности данных.

Пример: При добавлении данных в таблицу, если установлено ограничение на уникальность значений (UNIQUE), транзакция должна завершиться с ошибкой, если нарушает это ограничение. База данных останется в согласованном состоянии.

Изолированность (Isolation)

Одновременные транзакции не влияют друг на друга. Изменения, внесённые одной транзакцией, не видны другим до её завершения.

Пример: Если один пользователь редактирует данные заказа, а другой пытается просмотреть те же данные, то второй пользователь не увидит изменения, пока первая транзакция не завершится.

Долговечность (Durability)

После фиксации транзакции (COMMIT), все её изменения сохраняются в базе данных и остаются доступными даже при сбое системы.

Пример: Если транзакция завершилась успешно и зафиксировала изменения, то после перезапуска системы данные останутся сохранёнными.

Примеры использования ACID-свойств

Атомарность: В интернет-магазине при оформлении заказа на несколько товаров в корзине, если один из товаров отсутствует, вся операция должна быть отменена.

Согласованность: Если существует правило, что общая сумма транзакций не должна превышать баланс счёта, согласованность гарантирует, что система не позволит превысить этот лимит.

Изолированность: В банке несколько сотрудников могут работать с одним и тем же клиентом, но каждый из них должен видеть согласованные данные только после завершения всех транзакций.

Долговечность: После того как заказ в интернет-магазине был подтверждён, данные должны сохраняться в базе даже при отключении электричества или сбое системы.

ACID-свойства помогают обеспечить надёжность транзакций, предотвращая потерю данных и гарантируя их целостность.

3. Управление транзакциями.

Управление транзакциями позволяет контролировать выполнение операций с базой данных и обеспечивать целостность данных в случае ошибок.

Далее рассмотрим основные команды для работы с транзакциями.

START TRANSACTION

Эта команда используется для начала транзакции. Все изменения, которые будут сделаны после этой команды, временно сохраняются и не будут видны другим пользователям, пока транзакция не завершится.

Пример:

```
START TRANSACTION;
UPDATE accounts SET balance = balance - 100 WHERE account_id = 1;
UPDATE accounts SET balance = balance + 100 WHERE account_id = 2;
```

START TRANSACTION;

Эта команда начинает новую транзакцию. Все изменения, сделанные после неё, не будут зафиксированы (и не будут видны другим пользователям базы данных), пока не будет выполнена команда COMMIT. Это гарантирует, что если какая-либо из операций в транзакции не удастся, можно будет откатить изменения, вернув базу данных в исходное состояние с помощью ROLLBACK.

UPDATE accounts SET balance = balance - 100 WHERE account_id = 1;

Эта команда уменьшает баланс на счёте с account_id = 1 на 100 единиц. В данном случае, с указанного счёта происходит списание средств.

UPDATE accounts SET balance = balance + 100 WHERE account_id = 2;

Эта команда увеличивает баланс на счёте с account_id = 2 на 100 единиц. Это действие завершает процесс перевода средств — деньги, списанные с первого счёта, добавляются на второй.

Пример сценария:

Клиент переводит 100 единиц денег со своего счёта (account_id = 1) на счёт другого клиента (account_id = 2).

Сначала деньги списываются с первого счёта (100 единиц).

Затем они добавляются ко второму счёту.

COMMIT

Команда фиксирует все изменения, выполненные в рамках транзакции, и сохраняет их в базе данных. После выполнения COMMIT изменения становятся видны всем пользователям, и транзакцию уже нельзя откатить.

Пример:

```
COMMIT;
```

В результате все изменения, сделанные в транзакции (например, переводы средств), будут зафиксированы.

ROLLBACK

Эта команда откатывает транзакцию, отменяя все изменения, сделанные после её начала. Используется в случае ошибки или необходимости отменить операции.

Пример:

```
ROLLBACK;
```

Это вернёт базу данных в состояние, которое было до начала транзакции.

SAVEPOINT

Команда создаёт промежуточную точку внутри транзакции, к которой можно вернуться в случае необходимости. Это позволяет откатывать транзакцию не полностью, а до определённого момента.

Пример:

```
SAVEPOINT point1;
UPDATE accounts SET balance = balance - 50 WHERE account_id = 1;
ROLLBACK TO point1;
```

В данном примере после создания SAVEPOINT point1, если операция изменения баланса на 50 не проходит, можно откатить транзакцию только до этой точки, не отменяя других изменений.

Примеры использования команд.

Перевод денег между счетами.

Представьте, что необходимо перевести деньги между двумя счетами. Если какой-то шаг в процессе перевода не удаётся (например, один из счетов заблокирован), изменения должны быть отменены:

```
START TRANSACTION;
UPDATE accounts SET balance = balance - 500 WHERE account_id = 1;
UPDATE accounts SET balance = balance + 500 WHERE account_id = 2;
COMMIT; -- зафиксировать изменения
```

Обновление заказа в интернет-магазине

В случае, если обновление товара или заказчика сталкивается с проблемой, необходимо отменить все изменения:

```
START TRANSACTION;
UPDATE orders SET status = 'shipped' WHERE order_id = 123;
UPDATE inventory SET quantity = quantity - 1 WHERE product_id = 456;

IF (ошибка_с_товаром) THEN
   ROLLBACK;
ELSE
   COMMIT;
```

Использование SAVEPOINT

Представим, что нужно изменить информацию о нескольких клиентах, но изменения для одного клиента могут быть необязательными. В этом случае можно использовать точку сохранения:

```
START TRANSACTION;
UPDATE customers SET balance = balance - 100 WHERE customer_id = 1;
SAVEPOINT sp1;
UPDATE customers SET balance = balance - 100 WHERE customer_id = 2;
-- Если возникает ошибка для customer_id = 2
ROLLBACK TO sp1; -- Откатываем только изменения для второго клиента
COMMIT;
```

4. Триггеры как механизм обеспечения целостности данных.

Триггер — это специальная программа (или процедура) в базе данных, которая автоматически выполняется (срабатывает) при возникновении определённых событий, таких как вставка, обновление или удаление данных в таблице. Триггеры помогают автоматизировать проверку и обработку данных, обеспечивая целостность и согласованность данных.

Назначение триггеров:

- Автоматическая валидация данных перед вставкой или обновлением.
- Поддержание целостности данных между связанными таблицами (например, каскадное обновление/удаление).
- Автоматическая обработка данных: запись истории изменений, ведение логов, расчёт значений и т.д.
- Снижение ошибок за счёт автоматической проверки бизнес-правил, предотвращая некорректные действия.

Типы триггеров:

- **BEFORE** — триггер срабатывает перед выполнением действия (вставки, обновления или удаления). Используется для проверки и изменения данных до того, как они попадут в таблицу. Например, можно использовать триггер BEFORE INSERT для проверки данных перед их вставкой или для автоматического изменения значения поля.
- **AFTER** — триггер срабатывает после выполнения действия. Обычно используется для дополнительной обработки данных, например, для ведения логов или каскадного обновления/удаления связанных данных.

Примеры использования триггеров

Пример 1: BEFORE INSERT — Проверка данных перед вставкой

Предположим, есть таблица employees, и мы хотим убедиться, что зарплата сотрудников всегда больше нуля.

NEW.salary — это новое значение, которое будет вставлено в таблицу. Если условие нарушается, триггер срабатывает и вызывает ошибку, предотвращая вставку некорректных данных.

```
CREATE TRIGGER check_salary_before_insert
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
    IF NEW.salary <= 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Salary must be greater than zero';
    END IF;
END;
```

Пример 2: BEFORE UPDATE — Автоматическое изменение данных перед обновлением

Допустим, в таблице products необходимо автоматически обрезать пробелы в названии продукта перед его обновлением.

Триггер автоматически удалит пробелы в начале и конце строки перед обновлением значения в столбце name.

```
CREATE TRIGGER trim_product_name_before_update
BEFORE UPDATE ON products
FOR EACH ROW
BEGIN
    SET NEW.name = TRIM(NEW.name);
END;
```

Пример 3: AFTER INSERT — Ведение логов после вставки данных

В этом примере создадим триггер для ведения журнала операций после добавления нового заказа в таблицу orders.

NEW.order_id — это значение идентификатора нового заказа. После добавления нового заказа триггер автоматически добавляет запись в таблицу order_logs с информацией о том, что была выполнена операция вставки.

```
CREATE TRIGGER log_after_insert
AFTER INSERT ON orders
FOR EACH ROW
BEGIN
    INSERT INTO order_logs (order_id, action, action_date)
    VALUES (NEW.order_id, 'INSERT', NOW());
END;
```

Пример 4: AFTER DELETE — Каскадное удаление данных

Предположим, есть две таблицы: customers и orders, где каждому клиенту соответствует несколько заказов. Если клиента удаляют, нужно автоматически удалить все его заказы.

OLD.customer_id — это идентификатор клиента, который был удалён. После удаления записи в таблице customers триггер автоматически удаляет все заказы этого клиента из таблицы orders.

```
CREATE TRIGGER delete_orders_after_customer_delete
AFTER DELETE ON customers
FOR EACH ROW
BEGIN
    DELETE FROM orders WHERE customer_id = OLD.customer_id;
END;
```

Преимущества использования триггеров:

- Автоматизация — триггеры автоматически выполняют нужные действия без участия пользователя.
- Целостность данных — обеспечивают соблюдение бизнес-правил, предотвращают некорректные операции и поддерживают консистентность данных.
- Производительность — выполнение триггеров на уровне базы данных быстрее, чем выполнение аналогичных проверок в приложении.

Недостатки триггеров:

- Сложность отладки — из-за автоматического выполнения триггеров ошибки и сбои могут быть сложнее обнаружить.
- Потенциальное замедление работы — при большом количестве триггеров на таблицу каждая операция может стать медленнее, особенно если триггеры выполняют сложные действия.
- Скрытая логика — триггеры могут содержать логику, которая не очевидна для разработчиков, работающих с приложением.

5. Заключение.

Производительность

Домашнее задание:

1. Повторить материал лекции.

Контрольные вопросы:

- Что такое транзакция в контексте работы с базой данных?
- Опишите каждое из ACID-свойств транзакций и приведите примеры.
- Какое значение имеет атомарность (Atomicity) для целостности данных?
- Что происходит, если нарушается одно из условий согласованности (Consistency) в транзакции?
- Какую роль играет изолированность (Isolation) при одновременном выполнении нескольких транзакций?
- Что гарантирует долговечность (Durability) в транзакциях?
- Объясните процесс управления транзакциями в MySQL, начиная с команды START TRANSACTION и заканчивая COMMIT и ROLLBACK.
- Для чего используется команда SAVEPOINT, и приведите пример её использования.
- Какие задачи решают триггеры в базе данных?
- Как триггеры помогают поддерживать целостность данных между связанными таблицами?

Список литературы:

1. В. Ю. Кара-ушанов SQL — язык реляционных баз данных
2. А. Б. ГРАДУСОВ. Введение в технологию баз данных
3. А.Мотеев. Уроки MySQL

Материалы лекций:

https://github.com/ShViktor72/Education

Обратная связь:

colledge20education23@gmail.com