# Тема №9 – Транзакции, блоки команд.

Транзакции — это фундаментальное понятие во всех СУБД. Суть транзакции в том, что она объединяет последовательность действий в одну операцию "всё или ничего". Промежуточные состояния внутри последовательности не видны другим транзакциям, и, если что-то помешает успешно завершить транзакцию, ни один из результатов этих действий не сохранится в базе данных.

Например, рассмотрим базу данных банка, в которой содержится информация о счетах клиентов, а также общие суммы по отделениям банка. Предположим, что мы хотим перевести 300 долларов со счёта Джозефа на счёт Дио. Простоты ради, соответствующие SQL-команды можно записать так:

UPDATE accounts SET balance = balance - 300.00

WHERE name = 'Josef';

UPDATE branches SET balance = balance - 300.00

WHERE name = (SELECT branch\_name FROM accounts WHERE name = 'Josef');

UPDATE accounts SET balance = balance + 300.00

WHERE name = 'Dio';

UPDATE branches SET balance = balance + 300.00

WHERE name = (SELECT branch\_name FROM accounts WHERE name = 'Dio');

Точное содержание команд здесь не важно, важно лишь то, что для выполнения этой довольно простой операции потребовалось несколько отдельных действий. При этом с точки зрения банка необходимо, чтобы все эти действия выполнились вместе, либо не выполнились совсем. Если Дио получит 300 долларов, но они не будут списаны со счёта Джозефа, объяснить это сбоем системы определённо не удастся. И наоборот, Джозеф вряд ли будет доволен, если она переведёт деньги, а до Дио они не дойдут. Нам нужна гарантия, что, если что-то помешает выполнить операцию до конца, ни одно из действий не оставит следа в базе данных. И мы получаем эту гарантию, объединяя действия в одну транзакцию. Говорят, что транзакция атомарна: с точки зрения других транзакций она либо выполняется и фиксируется полностью, либо не фиксируется совсем.

Нам также нужна гарантия, что после завершения и подтверждения транзакции системой баз данных, её результаты в самом деле сохраняются и не будут потеряны, даже если вскоре произойдёт авария. Например, если мы списали сумму и выдали её Дио, мы должны исключить возможность того, что сумма на его счёте восстановится, как только он выйдет за двери банка. Транзакционная база данных гарантирует, что все изменения записываются в постоянное хранилище (например, на диск) до того, как транзакция будет считаться завершённой.

Другая важная характеристика транзакционных баз данных тесно связана с атомарностью изменений: когда одновременно выполняется множество транзакций, каждая из них не видит незавершённые изменения, произведённые другими. Например, если одна транзакция подсчитывает баланс по отделениям, будет неправильно, если она посчитает расход в отделении Джозефа, но не учтёт приход в отделении Дио, или наоборот. Поэтому свойство транзакций "всё или ничего" должно определять не только, как изменения сохраняются в базе данных, но и как они видны в процессе работы. Изменения, производимые открытой транзакцией, невидимы для других транзакций, пока она не будет завершена, а затем они становятся видны все сразу.

В Postgres транзакция определяется набором SQL-команд, окружённым командами BEGIN и COMMIT. Таким образом, наша банковская транзакция должна была бы выглядеть так:

BEGIN;

UPDATE accounts SET balance = balance - 300.00

WHERE name = 'Josef';

...

COMMIT;

Если в процессе выполнения транзакции мы решим, что не хотим фиксировать её изменения (например, потому что оказалось, что баланс Джозефа стал отрицательным), мы можем выполнить команду ROLLBACK вместо COMMIT, и все наши изменения будут отменены.

Если вы не вставите команду BEGIN, то каждый отдельный оператор будет неявно окружён командами BEGIN и COMMIT (в случае успешного завершения). Группу операторов, окружённых командами BEGIN и COMMIT иногда называют блоком транзакции.

Некоторые клиентские библиотеки добавляют команды BEGIN и COMMIT автоматически и неявно создают за вас блоки транзакций.

Операторами в транзакции можно также управлять на более детальном уровне, используя точки сохранения. Точки сохранения позволяют выборочно отменять некоторые части транзакции и фиксировать все остальные. Определив точку сохранения с помощью SAVEPOINT, при необходимости вы можете вернуться к ней с помощью команды ROLLBACK TO. Все изменения в базе данных, произошедшие после точки сохранения и до момента отката, отменяются, но изменения, произведённые ранее, сохраняются.

Когда вы возвращаетесь к точке сохранения, она продолжает существовать, так что вы можете откатываться к ней несколько раз. С другой стороны, если вы уверены, что вам не придётся откатываться к определённой точке сохранения, её можно удалить, чтобы система высвободила ресурсы. Помните, что при удалении или откате к точке сохранения все точки сохранения, определённые после неё, автоматически уничтожаются.

Всё это происходит в блоке транзакции, так что в других сеансах работы с базой данных этого не видно. Совершённые действия становятся видны для других сеансов все сразу, только когда вы фиксируете транзакцию, а отменённые действия не видны вообще никогда.

Вернувшись к базе данных, предположим, что мы списываем 100 долларов со счёта Джозефа, добавляем их на счёт Дио, и вдруг оказывается, что деньги нужно было перевести Джонни. В данном случае мы можем применить точки сохранения:

BEGIN;

UPDATE accounts SET balance = balance - 300.00

WHERE name = 'Josef';

SAVEPOINT my\_savepoint;

UPDATE accounts SET balance = balance + 300.00

WHERE name = 'Dio';

-- ошибочное действие... забыть его и использовать счёт Уолли

ROLLBACK TO my\_savepoint;

UPDATE accounts SET balance = balance + 300.00

WHERE name = ' Johnny';

COMMIT;

Этот пример, конечно, несколько надуман, но он показывает, как можно управлять выполнением команд в блоке транзакций, используя точки сохранения. Более того, ROLLBACK TO — это единственный способ вернуть контроль над блоком транзакций, оказавшимся в прерванном состоянии из-за ошибки системы, не считая возможности полностью отменить её и начать снова.

Стандарт SQL определяет четыре уровня изоляции транзакций. Наиболее строгий из них — сериализуемый, определяется одним абзацем, говорящем, что при параллельном выполнении несколько сериализуемых транзакций должны гарантированно выдавать такой же результат, как если бы они запускались по очереди в некотором порядке. Остальные три уровня определяются через описания особых явлений, которые возможны при взаимодействии параллельных транзакций, но не допускаются на определённом уровне. Как отмечается в стандарте, из определения сериализуемого уровня вытекает, что на этом уровне ни одно из этих явлений невозможно. Стандарт описывает следующие особые условия, недопустимые для различных уровней изоляции:

**«грязное» чтение** - транзакция читает данные, записанные параллельной незавершённой транзакцией.

**неповторяемое чтение** - транзакция повторно читает те же данные, что и раньше, и обнаруживает, что они были изменены другой транзакцией (которая завершилась после первого чтения).

**фантомное чтение** - транзакция повторно выполняет запрос, возвращающий набор строк для некоторого условия, и обнаруживает, что набор строк, удовлетворяющих условию, изменился из-за транзакции, завершившейся за это время.

**аномалия сериализации** - результат успешной фиксации группы транзакций оказывается несогласованным при всевозможных вариантах исполнения этих транзакций по очереди.

Таблица №5 – Уровни изоляции транзакций

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Уровень изоляции** | **«Грязное» чтение** | **Неповторяемое чтение** | **Фантомное чтение** | **Аномалия сериализации** |
| Read uncommited (Чтение незафиксированных данных) | Допускается, но не в PG | Возможно | Возможно | Возможно |
| Read committed (Чтение зафиксированных данных) | Невозможно | Возможно | Возможно | Возможно |
| Repeatable read (Повторяемое чтение) | Невозможно | Невозможно | Допускается, но не в PG | Возможно |
| Serializable (Сериализуемость) | Невозможно | Невозможно | Невозможно | Невозможно |

В PostgreSQL вы можете запросить любой из четырёх уровней изоляции транзакций, однако внутри реализованы только три различных уровня, то есть режим Read Uncommitted в PostgreSQL действует как Read Committed.

В этой таблице также показано, что реализация Repeatable Read в PostgreSQL не допускает фантомное чтение. Стандарт SQL допускает возможность более строгого поведения: четыре уровня изоляции определяют только, какие особые условия не должны наблюдаться, но не какие обязательно должны.

**Read Committed** — уровень изоляции транзакции, выбираемый в PostgreSQL по умолчанию. В транзакции, работающей на этом уровне, запрос SELECT (без предложения FOR UPDATE/SHARE) видит только те данные, которые были зафиксированы до начала запроса; он никогда не увидит незафиксированных данных или изменений, внесённых в процессе выполнения запроса параллельными транзакциями. По сути запрос SELECT видит снимок базы данных в момент начала выполнения запроса. Однако SELECT видит результаты изменений, внесённых ранее в этой же транзакции, даже если они ещё не зафиксированы. Также заметьте, что два последовательных оператора SELECT могут видеть разные данные даже в рамках одной транзакции, если какие-то другие транзакции зафиксируют изменения после запуска первого SELECT, но до запуска второго.

В режиме Repeatable Read видны только те данные, которые были зафиксированы до начала транзакции, но не видны незафиксированные данные и изменения, произведённые другими транзакциями в процессе выполнения данной транзакции. Это самое строгое требование, которое стандарт SQL вводит для этого уровня изоляции, и при его выполнении предотвращаются все вышеописанные явления, за исключением аномалий сериализации. Как было сказано выше, это не противоречит стандарту, так как он определяет только минимальную защиту, которая должна обеспечиваться на каждом уровне изоляции.

Для выбора нужного уровня изоляции транзакций используется команда SET TRANSACTION. Синтаксис команды:

SET TRANSACTION режим\_транзакции [, ...]

SET TRANSACTION SNAPSHOT id\_снимка

SET SESSION CHARACTERISTICS AS TRANSACTION режим\_транзакции [, ...]

Где режим\_транзакции может быть следующим:

ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }

READ WRITE | READ ONLY

[ NOT ] DEFERRABLE

Команда SET TRANSACTION устанавливает характеристики текущей транзакции. На последующие транзакции она не влияет. SET SESSION CHARACTERISTICS устанавливает характеристики транзакции по умолчанию для последующих транзакций в рамках сеанса. Заданные по умолчанию характеристики затем можно переопределить для отдельных транзакций командой SET TRANSACTION.

К характеристикам транзакции относится уровень изоляции транзакции, режим доступа транзакции (чтение/запись или только чтение) и допустимость её откладывания. В дополнение к ним можно выбрать снимок, но только для текущей транзакции, не для сеанса по умолчанию.

Уровень изоляции транзакции определяет, какие данные может видеть транзакция, когда параллельно с ней выполняются другие транзакции.

Таблица №6 – Опции команды SET TRANSACTION ISOLATION LEVEL

|  |  |
| --- | --- |
| **Опция** | **Описание** |
| READ COMMITTED | Оператор видит только те строки, которые были зафиксированы до начала его выполнения. Этот уровень устанавливается по умолчанию. |
| REPEATABLE READ | Все операторы текущей транзакции видят только те строки, которые были зафиксированы перед первым запросом на выборку или изменение данных, выполненным в этой транзакции. |
| SERIALIZABLE | Все операторы текущей транзакции видят только те строки, которые были зафиксированы перед первым запросом на выборку или изменение данных, выполненным в этой транзакции. Если наложение операций чтения и записи параллельных сериализуемых транзакций может привести к ситуации, невозможной при последовательном их выполнении (когда одна транзакция выполняется за другой), произойдёт откат одной из транзакций с ошибкой serialization\_failure. |

В стандарте SQL определён ещё один уровень, READ UNCOMMITTED. В PostgreSQL уровень READ UNCOMMITTED обрабатывается как READ COMMITTED.

Уровень изоляции транзакции нельзя изменить после выполнения первого запроса на выборку или изменение данных (SELECT, INSERT, DELETE, UPDATE, FETCH или COPY) в текущей транзакции. Режим доступа транзакции определяет, будет ли транзакция только читать данные или будет и читать, и писать. По умолчанию подразумевается чтение/запись. В транзакции без записи запрещаются следующие команды SQL: INSERT, UPDATE, DELETE и COPY FROM, если только целевая таблица не временная; любые команды CREATE, ALTER и DROP, а также COMMENT, GRANT, REVOKE, TRUNCATE; кроме того, запрещаются EXPLAIN ANALYZE и EXECUTE, если команда, которую они должны выполнить, относится к вышеперечисленным. Это высокоуровневое определение режима только для чтения, которое в принципе не исключает запись на диск.

Свойство DEFERRABLE оказывает влияние, только если транзакция находится также в режимах SERIALIZABLE и READ ONLY. Когда для транзакции установлены все три этих свойства, транзакция может быть заблокирована при первой попытке получить свой снимок данных, после чего она сможет выполняться без дополнительных усилий, обычных для режима SERIALIZABLE, и без риска привести к сбою сериализации или пострадать от него. Этот режим подходит для длительных операций, например, для построения отчётов или резервного копирования.

Если команде SET TRANSACTION не предшествует START TRANSACTION или BEGIN, она выдаёт предупреждение и больше ничего не делает. Без SET TRANSACTION можно обойтись, задав требуемые режимы\_транзакции в операторах BEGIN или START TRANSACTION. Но для SET TRANSACTION SNAPSHOT такой возможности не предусмотрено.