Национальный исследовательский ядерный университет «МИФИ» (Московский Инженерно–Физический Институт)
Кафедра №42 «Криптология и кибербезопасность»

Лабораторная работа №2-2: «Транзакции. Изоляция транзакций»

Цель работы

Изучение механизмов базы данных, связанных с обеспечением целостности информации при одновременной работе многих пользователей и при возникновении сбоев.

Подготовка к работе

Как правило, в основе современной клиент-серверной СУБД лежит механизм отслеживания транзакций. Транзакция представляет собой атомарную группу операций. Это означает, что группа SQL-инструкций, объединённых в транзакцию, не может быть выполнена частично. В случае, если во время выполнения транзакции происходит сбой или ошибка, никакие данные, затронутые этой транзакцией не будут закреплены в базе данных. Очень часто при работе с данными эта гарантия является необходимой. Например, ситуация, когда списание средств с одного счёта было успешно выполнено, а зачисление на другой не произошло из-за сбоя, недопустима.

Первым — очевидным — назначением механизма транзакций является обеспечение целостности в случае ошибки или сбоя. Если в ходе инициализации СУБД обнаружит незавершённые транзакции (а отметка об успешном завершении транзакции ставится в самом конце), она предпримет действия по устранению эффекта этой транзакции и возвращению базы данных к исходному (на момент начала транзакции) состоянию. Также, при возникновении непредвиденной ошибки, пользовательский сценарий может сам выполнить отмену (откат; ROLLBACK) транзакции и вернуть данные в нетронутый вид.

Вторым, более сложным применением для механизма транзакций, является изоляция действий одновременно работающих пользователей друг от друга. Целью реализации СУБД в этом случае является предоставление корректных, согласованных данных всем пользователям при минимизации времени блокировок и ожидания.

Что касается согласованности: если вернуться к примеру выше, ситуация с тем, что списание средств уже произошло, а зачисление — ещё нет, недопустима не только с точки зрения возможного отказа. Если в этот момент параллельно работающий пользователь будет работать с данными по списаниям и зачислениям, он может он может получить некорректные результаты. Поэтому результаты незавершённой транзакции другого пользователя не являются частью базы данных до тех пор, пока она не будет полностью завершена. Это значит, что даже если пользователь А успел выполнить списание, но не успел выполнить зачисление, пользователь Б не увидит ни того, ни другого, пока транзакция не будет закрыта. Затем ему будет предоставлена база данных, где обе операции уже завершены.

«Защита» пользователя от изменений, внесённых другими пользователями во время его работы, называется «изоляция транзакций». И иногда описанного выше уровня изоляции недостаточно. Стандарт ISO SQL регламентирует 4 различных уровня изоляции транзакций. Первый называется READ UNCOMMITTED. Или, другими словами, «отсутствие изоляции». На этом уровне один пользователь может читать изменения, которые внесли транзакции других пользователей, сразу после того, как эти изменения были внесены (не дожидаясь окончания транзакции), — это событие называется «грязное чтение» (dirty read). Как правило (= Oracle Database и PostgreSQL) этот уровень изоляции не поддерживается и полезен только при объяснении теоретических основ изоляции транзакций.

Второй уровень называется READ COMMITTED, и в Oracle Database и в PostgreSQL принят как уровень по умолчанию. Он работает так, как описано в примере выше: до тех пор, пока пользователь не завершит транзакцию (COMMIT), внесённые им изменения не являются частью базы данных и не видны для других пользователей. Как только транзакция завершается, данные появляются в базе данных.

Однако, такого уровня изоляции может быть недостаточно. Например, при составлении квартального отчёта недопустимо, чтобы новые данные появлялись в базе данных во время вычислений. При выполнении такой задачи желательно «заморозить» базу данных на определённый момент времени и не допускать новых поступлений информации до завершения работы данных транзакций. Поэтому дополнительно вводятся ещё два уровня изоляции.

Третий уровень называется REPEATABLE READ. Его требования понятны из названия: когда бы в ходе транзакции ни была прочитана определённая запись, мы требуем, чтобы до конца транзакции чтение этой записи возвращало те же самые результаты. То есть: если информация в базе данных изменилась, мы не увидим изменений до окончания нашей транзакции, даже если изменяющие транзакции завершатся раньше. REPEATABLE READ является полумерой: он защищает от изменения существующих данных, но не защищает от появления новых. Появление новых данных в БД в ходе транзакции называется «фантомное чтение» (phantom read). Поэтому в Oracle Database этот уровень изоляции официально полностью не поддерживается, а PostgreSQL позволит начать транзакцию на этом уровне, но фактически будет применять более высокий уровень изоляции, не допуская фантомного чтения..

Четвёртый, наивысший уровень изоляции, называется SERIALIZABLE («сериализуемая транзакция»). В Стандарте он задан как такой уровень, при котором фактическое выполнение транзакций приводит к такому же результату, как выполнение всех этих транзакций последовательно, в том порядке, в котором они выполнялись. Находясь в сериализуемой транзакции, пользователь полностью изолирован от действий других пользователей до её окончания.

Преимущества, предоставляемые повышенным уровнем изоляции, сопровождаются соизмеримыми недостатками: поддержка сериализуемой транзакции требует существенно больших ресурсов (СУБД не может «забыть» об уже завершившихся транзакциях, потому что сериализуемая транзакция должна думать, что их ещё не было). Отдельный кошмар возникает при попытке изменить данные из сериализованной транзакции: так как она оперирует, возможно, уже не акутальными данными, изменение, которое она собирается внести в данные, может быть неверным. Поэтому при выполнении сериализуемых транзакций велик риск возникновения неисправимых ошибок, связанных с невозможностью внесения изменений и исчерпанием системных ресурсов. В обоих случаях придётся выполнять полный откат транзакции и начинать работу сначала.

Вышесказанное обусловило выбор уровня REPEATABLE READ как уровня изоляции транзакций по умолчанию. Необходимость повышения уровня до SERIALIZABLE есть далеко не всегда, а иногда неповторяющиеся и фантомные чтения даже желательны.

Все операции, выполняемые в клиент-серверных СУБД Oracle Database и PostgreSQL — проходят в рамках транзакции (разумеется, речь только об инструкциях DML). Если транзакция не начата вручную при помощи инструкции SET TRANSACTION, она будет начата автоматически. Для завершения транзакции можно использовать SQL-инструкции COMMIT (подтверждение) или ROLLBACK (откат). SQLite3 также поддерживает транзакции, но там они служат только для обеспечения целостности данных.

Важно: многие клиентские инструменты обладают достаточно неприятной для понимающих людей функцией “autocommit“: автоматически завершают транзакцию инструкцией COMMIT без ведома пользователя. Это может спасти начинающего пользователя; но того, кто владеет механизмом транзакций, может подвести. Некоторые инструменты могут даже выполнять autocommit после каждой группы инструкций или после каждой инструкции. Поведение используемой клиентской программы в плане autocommit важно знать.

Важно: все SQL-инструкции являются атомарными (и сериализуемыми) вне зависимости от состояния и режима транзакции. Нельзя получить несогласованные данные в рамках одного SELECT. Нельзя модифицировать только часть запрошенных записей с помощью UPGRADE. Любая SQL-инструкция либо выполняется полностью, либо не выполняется вообще. Поэтому отсутствует какой-либо смысл в оборачивании каждой инструкции в отдельную транзакцию.

Так как изменения в базе данных могут возникнуть только по завершении транзакции, время в базе

данных фактически измеряется в номерах транзакций. В PostgreSQL транзакции сокращённо обозначаются xact ("cross-action"?).

Важно: то, что данные незавершённой транзакции не являются частью базы данных официально, не значит, что они не вносятся в базу данных до момента завершения транзакции. Информация в БД будет модифицирована незамедлительно после того, как выполнена соответствующая SQL-инструкция. Затронутые данные будут помечены номером этой транзакции. Если эти данные понадобятся кому-то до завершения транзакции, серверный процесс обнаружит, что они были изменены активной транзакцией и СУБД использует системный журнал, чтобы восстановить данные для этого клиента в таком виде, в каком они были. Созданная таким образом копия данных называется CR-клон (CR = Consistent Read).

Такая практика введена не случайно: СУБД рассчитана на то, что большинство транзакций будет заканчиваться успешно (COMMIT), поэтому все изменения сразу вносятся в активную базу данных. Как только транзакция завершается, достаточно просто поставить отметку о её завершении. Поэтому операция COMMIT выполняется почти мгновенно, а ROLLBACK может занимать некоторое время.

Вопросы для самоконтроля и самостоятельного изучения

1. Что такое транзакция?
2. Как выключить поддержку транзакций?
3. Как задать уровень изоляции транзакции?
4. В какой момент может завершиться транзакция?
5. Что такое откат (ROLLBACK) транзакции?
6. Что такое подверждение (COMMIT) транзакции?
7. Что такое (SAVEPOINT) и как им воспользоваться?
8. Каким образом (SAVEPOINT) влияет на другие транзакции в базе данных?
9. Что такое грязное чтение (dirty read)?
10. Что такое фантомное чтение (phantom read)?
11. При каких условиях желательно использовать транзакцию (именно) на уровне изоляции SERIALIZABLE?
12. При каких условиях желательно использовать транзакцию (именно) на уровне изоляции REPEATABLE READ?
13. Что делает СУБД при получении инструкции COMMIT?
14. Что делает СУБД при получении инструкции ROLLBACK?
15. Что такое xid?
16. Что такое xact?
17. Что такое CR-клон?
18. Как серверный процесс СУБД определяет, что необходимо построить CR-клон?
19. Что значит «SQL-инструкции всегда атомарны»?
20. Почему механизм транзакций затрагивает только инструкции DML?
21. Почему транзакция на уровне изоляции SERIALIZABLE не может изменить запись даже после того, как другая активная транзакция, изменившая её, завершится?
22. Зачем нужна поддержка транзакций в однопользовательской SQLite3?
23. Что будет делать СУБД, если при загрузке обнаружит записи о незавершённых транзакциях?
24. Зачем номер последней активной транзакции хранится в каждой записи каждой таблицы?

Ход работы

1. Создать две сессии в Вашей базе данных. Начать транзакцию на уровне изоляции READ COMMITTED в одной из сессий. Изменить и добавить какие-либо данные. Пронаблюдать за изменениями из обеих сессий. Объяснить полученные результаты. Завершить транзакцию инструкцией ROLLBACK. Пронаблюдать за изменениями из обеих сессий. Объяснить полученные результаты. Внести изменения ещё раз и завершить транзакцию инструкцией COMMIT. Пронаблюдать за изменениями из обеих сессий. Объяснить полученные результаты.

2. Выполнить те же операции для уровней изоляции REPEATABLE READ и SERIALIZABLE. Объяснить различия;
3. Выполнить модификацию одной или нескольких записей в сессии А, не завершая транзакцию. Попробовать прочитать эти данные и внести изменения из сессии Б. Завершить транзакцию в сессии А и попробовать ещё раз. Объяснить результат;
4. То же, но сессия Б находится на уровне изоляции SERIALIZABLE. Объяснить различия;
5. Оформить отчёт.

Оформление отчёта

1. Титульный лист: название института, название лабораторной работы, имя, фамилия, номер группы, год,...
2. Стенограмма всех сессий работы с СУБД с комментариями и ответами на поставленные в Ходе работы вопросы;
3. Заключение: краткое описание проделанной работы;