Требования ACID - набор требований, которые обеспечивают сохранность данных

A (Atomicity) - Атомарность
C (Consistency) - Согласованность
I (Isolation) - Изолированность
D (Durability) - Надежность

Атомарность

Атомарность гарантирует, что каждая транзакция будет выполнена полностью или не выполнена совсем. Не допускаются промежуточные состояния.

Пример: при отправке денег со счёта 1 на счёт 2 должны пройти две транзакции: списание денег со счёта 1 и получение денег на счёт 2. Если на счёте 2 истек срок годности карты или она заблокирована, то деньги не придут на счёт, но благодаря атомарности они не спишутся со счёта 1 и не зависнут, т.к. эти оба запроса часть одной транзакции.

Согласованность

Транзакция, достигающая своего нормального завершения (EOT - end of transaction, завершение транзакции) и, тем самым, фиксирующая свои результаты, сохраняет согласованность базы данных. Другими словами, каждая успешная транзакция по определению фиксирует только допустимые результаты.

Это свойство вытекает из предыдущего. Благодаря тому, что транзакция не допускает промежуточных результатов, база остается консистентной. Есть такое определение транзакции: "Упорядоченное множество операций, переводящих базу данных из одного согласованного состояния в другое". То есть до выполнения операции и после база остается консистентной (согласованной).

Пример: юзер в системе заполняет карточку: ФИО, дата рождения, телефон (код страны, города и номер), адрес (тоже разбит на несколько полей). В БД у нас есть несколько таблиц: client, phone, address. Когда юзер нажмёт "сохранить" в БД отправится 3 запроса (1 действие юзера -> 3 запроса к БД).
Можно отправить 3 разных запроса, но лучше сделать одну транзакцию, внутри которой будут эти 3 запроса.
Атомарность гарантирует, что не получится такого, что адрес с телефоном сохранились, а сам клиент - нет. Это сделало бы базу неконсистентной, ведь у нас бы появились атрибуты, «висящие в воздухе», никому не принадлежащие. Что, в свою очередь, приведет к ошибкам в системе.

За консистентностью должен следить разработчик. Ведь это вопрос скорее бизнес-логики, чем технологий. Те же атрибуты, "висящие в воздухе" - это разработчик знает, что:

- если есть телефон в таблице phone
- он должен ссылаться на таблицу client

База об этом не знает ничего, если ей не рассказать. И она легко пропустит запрос "добавь в базу телефон без ссылки на клиента", если сам по себе запрос корректный, а разработчик не повесил на таблицу foreign key.
Разработчик пишет код, пошагово переводящий БД в нужное согласованное состояние и, если где-то посередине возникает ошибка, откатывает всю транзакцию. То есть можно после каждого шага делать запрос, проверяя какое-то поле.

Изолированность

Во время выполнения транзакции параллельные транзакции не должны оказывать влияния на её результат.

В многопользовательских системах, где одновременно к БД обращаются много человек, транзакции запускают в параллель - для ускорения работы системы.

К каким эффектам может привести параллельная работа двух транзакций?

1 эффект: "Потерянная запись"

Есть некий счет А, на котором лежит 500$.

Кассир 1 списал с него 300. На выходе получается 200.

Кассир 2 тоже решил обратиться к этому же счету, и записал туда 300, пока К1 еще не успел закрыть свою транзакцию. Так как первая транзакция не закрыта, сумма на счете до сих пор 500, получаем 500 + 300 = 800.

Итог - мы "потеряли запись" первого кассира, ведь на выходе у нас А = 800, хотя должно быть 500.

2 эффект: "Грязное чтение"

Есть некий счет А, на котором лежит 500$.

Кассир 1 списал с него 300. Потом передумал и сделал откат - на выходе остались те же 500.

Кассиру 2 понадобилась информация по этому счету и он ее считал до того, как К1 закрыл свою транзакцию.

Итог - второй кассир считал неверную сумму.

3 эффект: "Повторимое чтение"

Есть некие данные.

Кассир 1 строит отчет. Операции идут последовательно для каждой колонки. Система считала данные, записала в первую колонку (например, взяв минимум от них).

Кассир 2 влез в эту таблицу данных и изменил некоторые счета в ней.

У кассира 1 продолжается построение отчета. И во вторую колонку система считывает уже новые данные.

Итог - отчет построен на основании разных данных.

4 эффект: "Фантомы"

Есть некие данные.

Кассир 1 строит отчет. Операции идут последовательно для каждой колонки. Система считала данные, записала в первую колонку (например, взяв минимум от них).

Кассир 2 влез в эту таблицу данных и добавил новые счета/удалил некоторые старые.

У кассира 1 продолжается построение отчета. И во вторую колонку система считывает уже новые данные.

Итог - отчет построен на основании разных данных.

Разница между 3-им и 4-ым эффектами в том, что в одном случае данные изменяются, а во втором — добавляются/удаляются. То есть меняется ещё и их количество.

Как бороться с этими проблемами? Нужно изолировать транзакцию. Способов есть несколько, но основные - блокировки и версии.

Блокировки - это когда мы блокируем данные в базе. Можно заблокировать одну строку в таблице, а можно всю таблицу. Можно заблокировать данные на редактирование, а можно и на чтение тоже.

Версии - это когда внутри базы при каждом обновлении создается новая версия данных и сохраняется старая. Версионирование скрыто от разработчика, то есть мы не видим в базе никаких номеров версий и данных по ним. Просто пока транзакция, обновляющая запись, не покомитит свое изменение, остальные потребители читают старую версию записи и не блокируются.

Надёжность

Если юзер получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя. Обесточилась система, произошел сбой в оборудовании? На выполненную транзакцию это не повлияет.

Транзакции

Транзакция — набор операций по работе с базой данных, объединенных в одну атомарную пачку.

Одной операции всегда соответствует одна транзакция, но в рамках одной транзакции можно совершить несколько операций (например, несколько разных insert можно сделать, или изменить и удалить данные).

Чтобы отправить транзакцию к базе, нам нужно создать соединение с ней или переиспользовать уже существующее. Соединение называют также коннект (connection) - это просто труба, по которой отправляются запросы. У базы есть пул соединений - место, откуда можно взять любое и использовать, они там все свободные.

В некоторых системах транзакцию нужно открыть, в других она открывается сама. А вот закрыть ее нужно самостоятельно.

Варианты:

COMMIT - подтверждаем все внесенные изменения;

ROLLBACK - откатываем их;

Делая комит, мы заканчиваем одну бизнес-операцию, и возвращаем коннект в пул без открытой транзакции. То есть просто освобождаем трубу для других. Следующая бизнес-операция берет эту трубу и отправляет в нее свои операции. Поэтому важно сделать rollback, если изменения сохранять не надо. Если не откатить и вернуть соединение в пул, его возьмет кто-то другой и сделает коммит, как своих изменений, так и неоткаченных.

Следует не путать соединение с базой (коннект) и саму транзакцию. Коннект - это просто труба, операции (update, delete и т.д) мы посылаем по трубе, старт транзакции и commit /rollback — это группировка операций в одну атомарную пачку.