Севастопольский государственный университет
Кафедра «Информационные системы»

Управление данными

курс лекций

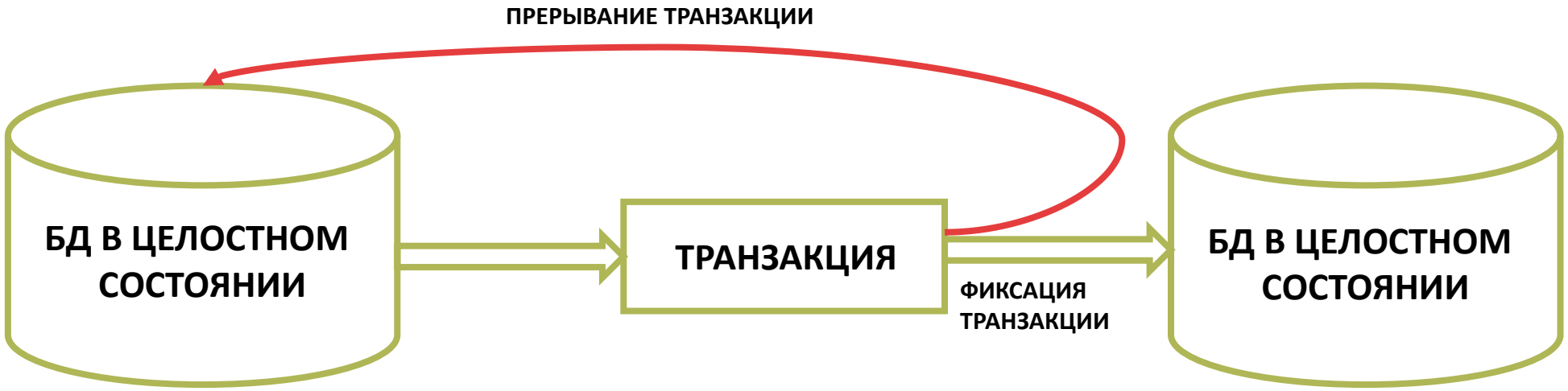
лектор:
ст. преподаватель кафедры ИС Абрамович А.Ю.

Лекция 6

Механизм транзакций и блокировки в базах данных. CAP-теорема

КОНЦЕПЦИЯ ТРАНЗАКЦИЙ

Под **транзакцией** понимается н**еделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными** (чтения, удаления, вставки, модификации), **приводящая к одному из двух возможных результатов**: либо последовательность выполняется, если все операторы правильные, либо вся транзакция прерывается, если хотя бы один оператор не может быть успешно выполнен. Обработка **транзакций гарантирует целостность информации в базе данных. Транзакция переводит базу данных из одного целостного состояния в другое.**



Что может быть названо транзакцией? Кем определяется, какая последовательность операций над базой данных составляет транзакцию? Однозначно именно **разработчик определяет**, какая последовательность операций составляет единое целое, то есть транзакцию.

Пример, который связан с принятием заказа в фирме на изготовление компьютера. Компьютер состоит из комплектующих, которые сразу резервируются за данным заказом в момент его формирования. Тогда т**ранзакцией будет вся последовательность операций, включающая следующие операции**:

- ввод нового заказа со всеми реквизитами заказчика;
- изменения состояния для всех выбранных комплектующих на складе на «занято» с привязкой их к определенному заказу;
- подсчет стоимости заказа с формированием платежного документа типа выставляемого счета к оплате;
- включение нового заказа в производство.

С точки зрения работника, это единая последовательность операций; если она будет прервана, то база данных потеряет свое целостное состояние.

Корректное поддержание транзакций является основой обеспечения целостности БД. Транзакции также **составляют основу изолированности в многопользовательских системах**, где с одной БД параллельно могут работать несколько пользователей или прикладных программ. Такую задачу СУБД принято называть **параллелизмом транзакций**.

Большинство выполняемых **действий производится в теле транзакций.** По умолчанию **каждая команда выполняется как самостоятельная транзакция.** При необходимости пользователь может явно указать ее начало и конец, чтобы иметь возможность включить в нее несколько команд.

При выполнении транзакции система управления базами данных должна придерживаться определенных правил обработки набора команд, входящих в транзакцию.

РЕАЛИЗАЦИЯ ТРАНЗАКЦИЙ

Реализация транзакций в СУБД PostgreSQL (Oracle, MySQL) основана на **многоверсионной модели (Multiversion Concurrency Control, MVCC).**

Позволяет осуществлять параллельное чтение и изменение записей (tuples) одних и тех же таблиц без блокировки этих таблиц. Чтобы иметь такую возможность, **данные из таблицы сразу не удаляются, а лишь помечаются как удаленные.** Изменение записей осуществляется путем маркировки этих записей как удаленных, и созданием новых записей с измененными полями. Таким образом, **история изменений одной записи сохраняется в базе данных и доступна для чтения другими транзакциями**. Этот способ хранения записей **позволяет параллельным процессам иметь неблокирующий доступ к записям, которые были удалены или изменены в параллельных незакрытых транзакциях**. У каждой записи в таблицы есть **системные скрытые поля xmin, xmax**.

- **xmin** - хранит номер транзакции, в которой **запись была создана**.
- **xmax** - хранит номер транзакции, в которой **запись была удалена или изменена**.

Модель **предполагает, что каждый SQL-оператор видит так называемый снимок данных (snapshot),** т. е. то согласованное состояние (версию) базы данных, которое она имела на определенный момент времени.

Снимок – это не физическая копия всей базы данных, это **несколько чисел, которые идентифицируют текущую транзакцию и те транзакции, которые уже выполнялись в момент начала текущей**. При этом *параллельно исполняемые транзакции*, даже вносящие изменения в базу данных, **не нарушают согласованности данных этого снимка.**

Такой результат достигается за счет того, что **когда параллельные транзакции изменяют одни и те же строки таблиц, тогда создаются отдельные версии этих строк**, доступные соответствующим транзакциям. Это позволяет **ускорить работу с базой данных**, но требует **больше дискового пространства** и оперативной памяти.

Важное следствие применения MVCC — **операции чтения никогда не блокируются операциями записи, а операции записи никогда не блокируются операциями чтения**.

УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ

Под **управлением транзакциями** понимается **способность управлять различными операциями над данными, которые выполняются внутри реляционной СУБД.** Прежде всего, имеется в виду **выполнение операторов *INSERT, UPDATE и DELETE***.

После успешного выполнения команд, заключенных в тело одной транзакции, **немедленного изменения данных не происходит**. Для окончательного **завершения транзакции существуют команды управления транзакциями**, с помощью которых можно либо сохранить в базе данных все изменения, произошедшие в ходе ее выполнения, либо полностью их отменить ***(COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION)***.

После завершения транзакции вся информация о произведенных изменениях хранится либо в специально выделенной оперативной памяти, либо во временной области отката в самой базе данных до тех пор, пока не будет выполнена одна из команд управления транзакциями. Затем все изменения или фиксируются в базе данных, или отбрасываются, а временная область отката освобождается.

Команды управления транзакциями

Команда COMMIT — это транзакционная команда, **используемая для сохранения изменений внесенных транзакцией в базу данных**. Команда COMMIT сохраняет все транзакции в базе данных с момента выполнения последней команды COMMIT или ROLLBACK.

Команда ROLLBACK — это транзакционная команда, **используемая для отмены транзакций, которые еще не были сохранены в базе данных**. Эта команда может использоваться только для отмены транзакций с момента выполнения последней команды COMMIT или ROLLBACK.

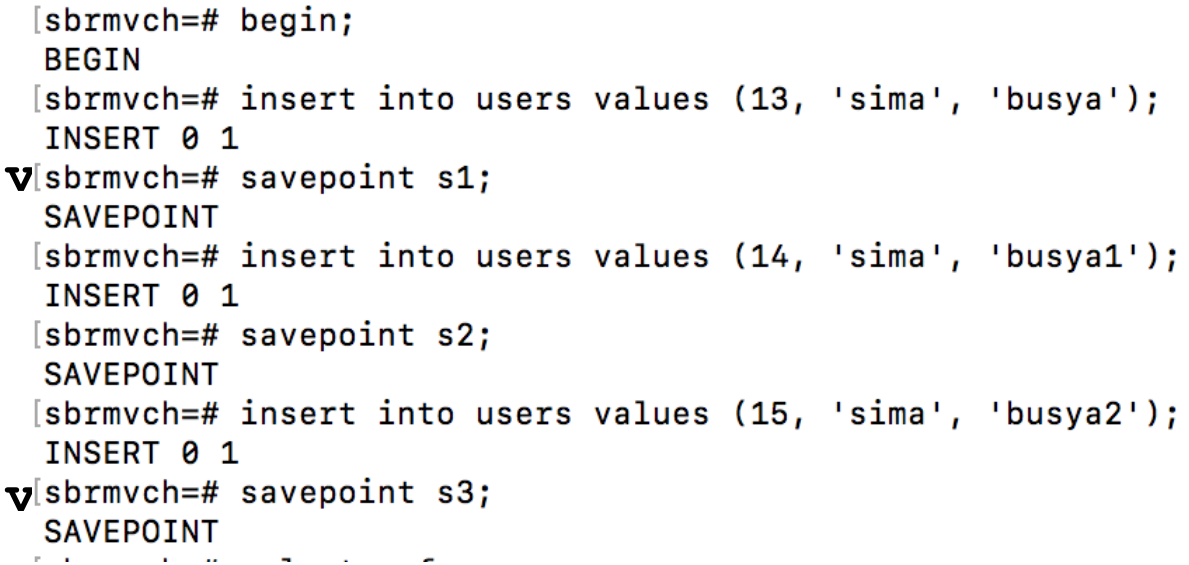
```
sbrmvch=# begin;
BEGIN
sbrmvch=# insert into users values (13, 'sima', 'busya');
INSERT 0 1
sbrmvch=# rollback;
ROLLBACK
```

```
sbrmvch=# select * from users;
 id |  name   |     profession
----+---------+--------------------
  1 | Bob     | QA
  2 | Camilo  | Front End developer
  3 | Billy   | Backend Developer
  4 | Alice   | Mobile Developer
  5 | Kate    | QA
  6 | Wayne   | DevOps
  7 | Tim     | Mobile Developer
  8 | Amigos  | QA
  9 | Sima    | QA
 10 | Kim     | QA
 11 | Kate    | QA
```

Команда SAVEPOINT

SAVEPOINT – это **точка транзакции, к которой можно вернуть транзакцию, не откатывая ее полностью.** Эта команда предназначена только для создания SAVEPOINT в других транзакционных операторах. Команда ROLLBACK используется для отмены группы транзакций до точки SAVEPOINT.

```
sbrmvch=# begin;
BEGIN
sbrmvch=# insert into users values (13, 'sima', 'busya');
INSERT 0 1
sbrmvch=# savepoint s1;
SAVEPOINT
sbrmvch=# insert into users v
'busya1');
INSERT 0 1
sbrmvch=# savepoint s2;
SAVEPOINT
sbrmvch=# insert into users v
'busya2');
INSERT 0 1
sbrmvch=# savepoint s3;
SAVEPOINT
```



```
sbrmvch=# select * from users;
 id |  name   |     profession
----+---------+---------------------
  1 | Bob     | QA
  2 | Camilo  | Front End developer
  3 | Billy   | Backend Developer
  4 | Alice   | Mobile Developer
  5 | Kate    | QA
  6 | Wayne   | DevOps
  7 | Tim     | Mobile Developer
  8 | Amigos  | QA
  9 | Sima    | QA
 10 | Kim     | QA
 11 | Kate    | QA
 13 | sima    | busya
 14 | sima    | busya1
 15 | sima    | busya2
(14 rows)
```

sbrmvch=# rollback to savepoint s2;
ROLLBACK

```
sbrmvch=# select * from users;
 id |  name   |     profession
----+---------+---------------------
  1 | Bob     | QA
  2 | Camilo  | Front End developer
  3 | Billy   | Backend Developer
  4 | Alice   | Mobile Developer
  5 | Kate    | QA
  6 | Wayne   | DevOps
  7 | Tim     | Mobile Developer
  8 | Amigos  | QA
  9 | Sima    | QA
 10 | Kim     | QA
 11 | Kate    | QA
 13 | sima    | busya
 14 | sima    | busya1
(13 rows)
```

```
sbrmvch=# rollback to savepoint s1;
ROLLBACK
```

```
sbrmvch=# rollback to savepoint s1;
ROLLBACK
sbrmvch=# select * from users;
 id |  name   |     profession
----+---------+--------------------
  1 | Bob     | QA
  2 | Camilo  | Front End developer
  3 | Billy   | Backend Developer
  4 | Alice   | Mobile Developer
  5 | Kate    | QA
  6 | Wayne   | DevOps
  7 | Tim     | Mobile Developer
  8 | Amigos  | QA
  9 | Sima    | QA
 10 | Kim     | QA
 11 | Kate    | QA
 13 | sima    | busya
(12 rows)

sbrmvch=# commit;
COMMIT
```

Команда RELEASE SAVEPOINT используется для **удаления созданной точки SAVEPOINT**. После того как SAVEPOINT будет удалена, больше нельзя использовать команду ROLLBACK для отмены транзакций, выполненных после последней SAVEPOINT.

Команда SET TRANSACTION может использоваться **для инициирования транзакции базы данных**. Эта команда используется для указания характеристик транзакций, которая задается после команды. Например, можно задать для транзакции режим только чтения или чтение и запись.

ЖУРНАЛ ТРАНЗАКЦИЙ

Реализация в СУБД **принципа сохранения промежуточных состояний, подтверждения или отката транзакции** обеспечивается специальным механизмом, для поддержки которого создается некоторая системная структура, называемая **журналом транзакций**.

Общей **целью журнализации** изменений баз данных является **обеспечение возможности восстановления согласованного состояния базы данных после любого сбоя.** Поскольку основой поддержания целостного состояния базы данных является механизм транзакций, журнализация и восстановление тесно связаны с понятием транзакции. **Общими принципами восстановления являются следующие:**

- результаты зафиксированных транзакций **должны быть сохранены в восстановленном состоянии базы данных**;
- результаты незафиксированных транзакций **должны отсутствовать в восстановленном состоянии базы данных.**

Это, собственно, и означает, что **восстанавливается последнее по времени согласованное состояние базы данных.**

Ситуации, при которых требуется производить восстановление состояния базы данных.

Индивидуальный откат транзакции. Этот откат должен быть применен в **следующих случаях**:

- стандартной ситуацией отката транзакции является ее **явное завершение оператором ROLLBACK**;
- **аварийное завершение работы прикладной программы**, которое логически эквивалентно выполнению оператора ROLLBACK, но физически имеет иной механизм выполнения;
- **принудительный откат транзакции в случае взаимной блокировки при параллельном выполнении транзакций**. В подобном случае для выхода из тупика данная транзакция может быть выбрана в качестве «жертвы» и принудительно прекращено ее выполнение ядром СУБД.

Восстановление после внезапной потери содержимого оперативной памяти (мягкий сбой). Такая ситуация может возникнуть в **следующих случаях**:

- при **аварийном выключении электрического питания**;
- при **возникновении неустранимого сбоя процессора** (например, срабатывании контроля оперативной памяти) и т. д. Ситуация характеризуется потерей той части базы данных, которая к моменту сбоя содержалась в буферах оперативной памяти.

Восстановление после поломки основного внешнего носителя базы данных (жесткий сбой). Эта ситуация при достаточно высокой надежности современных устройств внешней памяти может возникать сравнительно редко, но тем не менее СУБД должна быть в состоянии восстановить базу данных даже и в этом случае. Основой восстановления является архивная копия и журнал изменений базы данных.

ДЛЯ КАЖДОЙ ТРАНЗАКЦИИ ПОДДЕРЖИВАЕТСЯ ОТДЕЛЬНЫЙ ЛОКАЛЬНЫЙ ЖУРНАЛ ИЗМЕНЕНИЙ БАЗЫ ДАННЫХ ЭТОЙ ТРАНЗАКЦИЕЙ.

Такие журналы называются **локальными журналами.** Они используются **для индивидуальных откатов транзакций и могут поддерживаться в оперативной памяти**. Кроме того, поддерживается общий журнал изменений базы данных, используемый для восстановления состояния базы данных после **мягких и жестких сбоев**.

Этот подход позволяет быстро выполнять индивидуальные откаты транзакций, но **приводит к дублированию информации в локальных и общем журналах**.

ПОДДЕРЖАНИЕ ТОЛЬКО ОБЩЕГО ЖУРНАЛА ИЗМЕНЕНИЙ БАЗЫ ДАННЫХ, КОТОРЫЙ ИСПОЛЬЗУЕТСЯ И ПРИ ВЫПОЛНЕНИИ ИНДИВИДУАЛЬНЫХ ОТКАТОВ.

Общая структура журнала условно может быть представлена **в виде некоторого последовательного файла, в котором фиксируется каждое изменение БД, которое происходит в ходе выполнения транзакции**. Все транзакции имеют свои внутренние номера, поэтому в едином журнале транзакций фиксируются все изменения, проводимые всеми транзакциями.
Каждая запись в журнале транзакций помечается номером транзакции, к которой она относится, и значениями атрибутов, которые она меняет.

Для каждой транзакции в журнале фиксируется команда начала и завершения транзакции

ПЛОСКИЕ (КЛАССИЧЕСКИЕ, ТРАДИЦИОННЫЕ) ТРАНЗАКЦИИ

РАСПРЕДЕЛЕННЫЕ ТРАНЗАКЦИИ

ВЛОЖЕННЫЕ ТРАНЗАКЦИИ

Если **транзакция состоит из набора операций над объектами**, она называется **плоской (flat). Основное ограничение** плоских транзакций состоит в том, что **клиент не может воспользоваться промежуточными результатами транзакции**, например в случае ее прерывания.

Транзакции, выполняющиеся в составе других транзакций, принято называть **вложенными (nested).** Вложенные транзакции снимают некоторые ограничения плоских транзакций и могут выполняться параллельно для повышения производительности. **Транзакции, затрагивающие объекты, расположенные на разных машинах,** называют **распределенными**. Распределенная транзакция может быть как плоской, так и вложенной.

Для большей надежности **журнал транзакций часто дублируется системными средствами коммерческих СУБД**, именно поэтому **объем внешней памяти во много раз превышает реальный объем данных, которые хранятся в хранилище**.

Имеются **два альтернативных варианта ведения журнала транзакций**: протокол с отложенными обновлениями и протокол с немедленными обновлениями.

ACID-свойства транзакций

Разработано четыре правила, известные как требования ACID (Atomicity, Consistency, Isolation, Durability – неделимость, согласованность, изолированность, устойчивость), они гарантируют правильность и надежность работы системы.

Транзакция неделима в том смысле, что **представляет собой единое целое.**

Все ее компоненты либо имеют место, либо нет.

Не бывает частичной транзакции.

Если может быть **выполнена лишь часть транзакции, она отклоняется.**

Транзакция является согласованной, потому что **не нарушает бизнес-логику** и отношения между элементами данных.

Если хотя **бы одна транзакция нарушит целостность данных, то все остальные могут выдать неверные результаты.**

Транзакция всегда изолирована, поскольку ее **результаты самодостаточны.**

Они не зависят от предыдущих или последующих транзакций.

Это свойство называется сериализуемостью – транзакции в последовательности независимы.

Транзакция устойчива.

После своего завершения она сохраняется в системе, которую ничто не может вернуть в исходное состояние, т.е. **происходит фиксация транзакции**, означающая, что **ее действие постоянно** даже при сбое системы.

БЛОКИРОВКИ

Повышение эффективности работы при использовании небольших транзакций связано с тем, что при выполнении транзакции сервер накладывает ***на данные блокировки***.

Блокировкой называется **временное ограничение на выполнение некоторых операций обработки данных**. Блокировка может быть **наложена как на отдельную строку таблицы, так и на всю базу данных**. Управлением блокировками на сервере занимается менеджер блокировок, контролирующий их применение и разрешение конфликтов. ***Транзакции накладывают блокировки на данные, чтобы обеспечить выполнение требований ACID.*** Без использования блокировок несколько транзакций могли бы изменять одни и те же данные.

Блокировка представляет собой метод управления параллельными процессами, при котором объект БД не может быть модифицирован без ведома транзакции, т.е. происходит блокирование доступа к объекту со стороны других транзакций, чем исключается непредсказуемое изменение объекта. Различают **два вида** блокировки:

- **блокировка записи** – транзакция блокирует строки в таблицах таким образом, что запрос другой транзакции к этим строкам будет отменен ;
- **блокировка чтения** – транзакция блокирует строки так, что запрос со стороны другой транзакции на блокировку записи этих строк будет отвергнут, а на блокировку чтения – принят.

Решение проблемы параллельной обработки БД заключается в том, **что строки таблиц блокируются, а последующие транзакции, модифицирующие эти строки, отвергаются и переводятся в режим ожидания.** В связи со свойством сохранения целостности БД транзакции являются подходящими единицами изолированности пользователей. Если каждый сеанс взаимодействия с базой данных реализуется транзакцией, то пользователь начинает с того, что обращается к согласованному состоянию базы данных – состоянию, в котором она могла бы находиться, даже если бы пользователь работал с ней в одиночку.

В СУБД используют **протокол доступа к данным, позволяющий избежать проблемы параллелизма**. Его суть заключается **в следующем**:

- ***транзакция***, результатом действия которой на строку данных в таблице является ее извлечение, **обязана наложить блокировку чтения на эту строку**;
- ***транзакция***, предназначенная для модификации строки данных, **накладывает на нее блокировку записи**;
- если запрашиваемая **блокировка на строку отвергается** из-за уже имеющейся блокировки, то ***транзакция переводится в режим ожидания*** до тех пор, пока блокировка не будет снята;
- **блокировка записи сохраняется** вплоть до конца выполнения транзакции.

Если в системе управления базами данных не реализованы механизмы блокирования, **то при одновременном чтении и изменении одних и тех же данных несколькими пользователями могут возникнуть следующие проблемы одновременного доступа**:

- **потерянное обновление (lost update).** Когда разные транзакции одновременно изменяют одни и те же данные, то после фиксации изменений может оказаться, что одна транзакция перезаписала данные, обновленные и зафиксированные другой транзакцией;
- **«грязное» чтение (dirty read).** Транзакция читает данные, измененные параллельной транзакцией, которая еще не завершилась. Если эта параллельная транзакция в итоге будет отменена, тогда окажется, что первая транзакция прочитала данные, которых нет в системе;
- **неповторяющееся чтение (non-repeatable read).** При повторном чтении тех же самых данных в рамках одной транзакции оказывается, что другая транзакция успела изменить и зафиксировать эти данные. В результате тот же самый запрос выдает другой результат;
- **фантомное чтение (phantom read).** Транзакция выполняет повторную выборку множества строк в соответствии с одним и тем же критерием. В интервале времени между выполнением этих выборок другая транзакция добавляет новые строки и успешно фиксирует изменения. В результате при выполнении повторной выборки в первой транзакции может быть получено другое множество строк;
- **аномалия сериализации (serialization anomaly).** Результат успешной фиксации группы транзакций, выполняющихся параллельно, не совпадает с результатом ни одного из возможных вариантов упорядочения этих транзакций, если бы они выполнялись последовательно.

Для решения перечисленных проблем в специально разработанном стандарте определены уровни изоляции.

CAP-теорема

Теорема CAP (Consistency, Availability, Partition Tolerance) гласит, что **невозможно создать распределенную систему, которая бы одновременно обладала тремя свойствами**: **согласованностью** (Consistency), **доступностью** (Availability) и **устойчивостью к разделению** (Partition Tolerance).

СОГЛАСОВАННОСТЬ (CONSISTENCY)

определяет, что **каждый узел в распределенной системе должен иметь доступ к одному и тому же состоянию данных в любой момент времени**. Если система обладает данным свойством, то любые изменения данных, которые происходят в одном узле, должны быть немедленно доступны во всех остальных узлах системы.

ДОСТУПНОСТЬ (AVAILABILITY)

определяет, что **каждый запрос, отправленный к системе, должен получить ответ, даже если произошел сбой в системе**. Если система обладает свойством доступности, то каждый узел должен быть доступен для обработки запросов в любое время, даже если другие узлы системы недоступны.

УСТОЙЧИВОСТЬ К РАЗДЕЛЕНИЮ (PARTITION TOLERANCE)

определяет, что **система должна продолжать функционировать, даже если происходит разделение сети на две или более частей**. Если система обладает свойством устойчивости к разделению, то каждый узел должен продолжать работать независимо от других узлов, даже если эти узлы временно не могут связаться друг с другом.

Следствие 1

Одним из основных следствий CAP теоремы является то, что **распределенная система может обеспечить только два из трех свойств** - согласованность (Consistency), доступность (Availability) и устойчивость к разделению (Partition Tolerance). Это означает, что при разработке распределенных систем необходимо выбрать, какие свойства являются для данной системы наиболее важными.

Следствие 2

При проектировании распределенных систем **необходимо учитывать множество факторов, включая пропускную способность сети, нагрузку на узлы, объем данных** и многое другое. Необходимо принимать во внимание **возможные сбои в системе и различные виды атак на безопасность.** Для обеспечения требуемого уровня производительности, надежности и безопасности распределенной системы могут использоваться различные подходы, такие как репликация данных, шардинг, резервирование ресурсов и многое другое.

ДОСТУПНОСТЬ И СОГЛАСОВАННОСТЬ (CA)

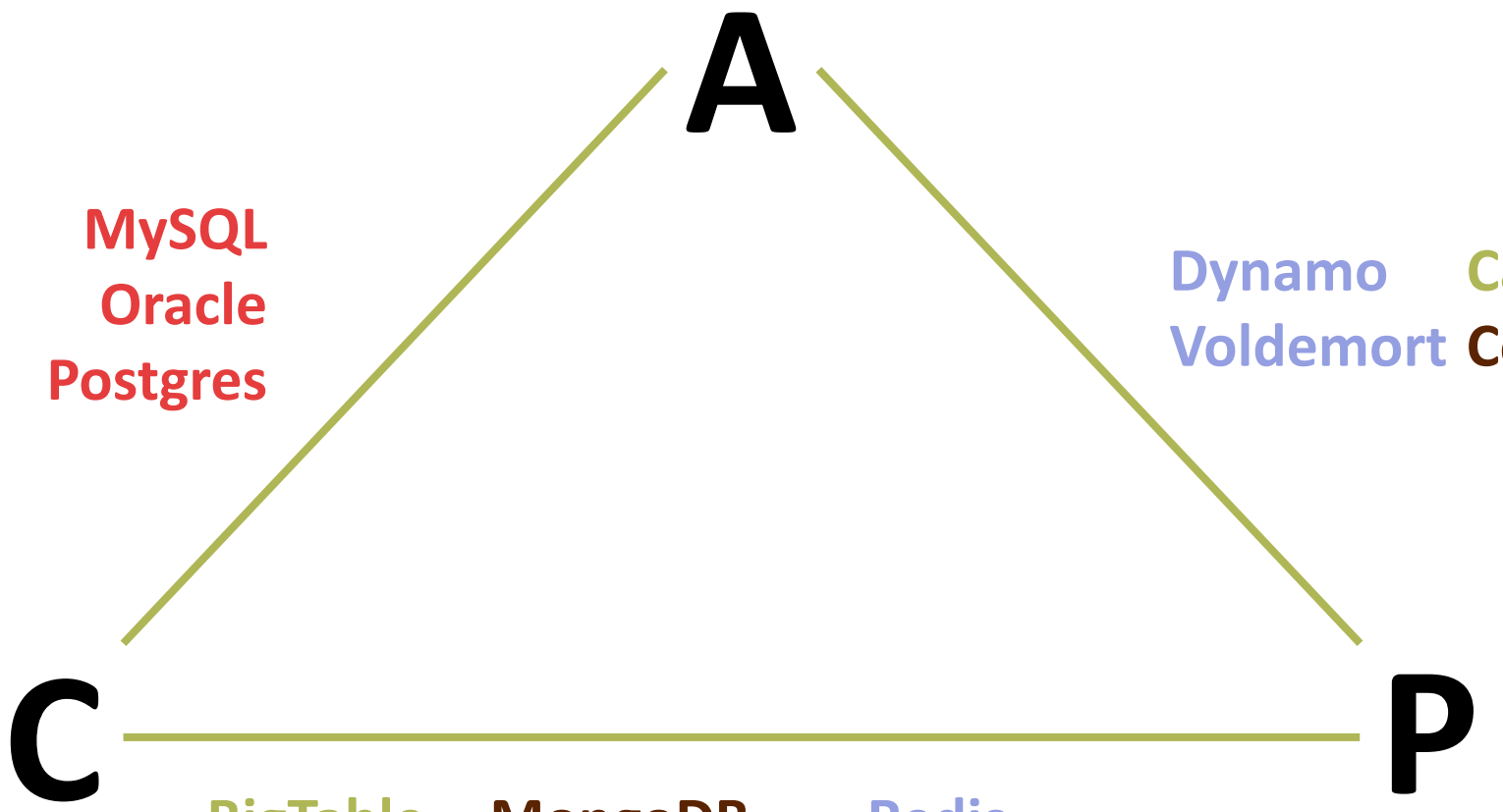
Для обеспечения идеальной **доступности** в распределенной системе необходимо, чтобы **каждый запрос был обработан успешно, без задержек и сбоев**. В то же время, для обеспечения идеальной **согласованности** необходимо, чтобы **все узлы в системе имели одинаковое представление о состоянии данных в системе.**

ДОСТУПНОСТЬ И УСТОЙЧИВОСТЬ К РАЗДЕЛЕНИЮ (AP)

Доступность означает, что каждый запрос к системе должен быть **обработан успешно без задержек и сбоев, вне зависимости от возможных сбоев узлов в системе. Устойчивость к разделению** означает, что система должна **продолжать работу при разделении сети на несколько частей**.

СОГЛАСОВАННОСТЬ И УСТОЙЧИВОСТЬ К РАЗДЕЛЕНИЮ (CP)

Пересечение согласованности (Consistency) и устойчивости к разделению (Partition Tolerance) в означает, что **распределенная система должна сохранять согласованность данных при возможном разделении сети на несколько частей**.



Модели данных:

Реляционная

Ключ-значение

Столбцовая

Документная