	A	B	C	D
1	Попытка сравнения сочетаний (с количеством свойств от 1 до 2, всегда меньше 3х) CAP теоремы для **РАСПРЕДЕЛЕННЫХ** систем на примерах СУБД и соответствующих use cases.			
2	**Особенность / Вариант CAP (в скобках частичная неполная реализация), где P – стойкость к разделению**	**(C)(A)**	**(C)P**	**(A)P**
3	**Consistency (Согласованность или точнее линеаризуемость) C** **Реплики на разных узлах никогда не противоречат друг другу.** **Все клиенты видят одинаковые данные вне зависимости от того, к какому узлу они подключены.**	Любой клиент при запросе к любому живому узлу всегда получает одинаковый ответ (с самыми свежими данными после commit, сделанном ранее на любом узле). Во всех вычислительных узлах в один момент времени данные не противоречат друг другу.		Не гарантирует полную согласованность, только eventually (weak) consistent. Может вернуть противоречивые неактуальные данные, которые пока ивестны на выбранном клиентом узле. А актуальные данные могут появиться на этом узле потом с задержкой.
4		Пока нет разделения сети, то во всех узлах данные согласованы и обеспечена доступность, но при этом нет устойчивости к распаду на секции. В случае распада для отсеченной ведомой секции исчезает одновременно и консистентность и доступность, а ведущий узел все же может остаться CA.	В каждый момент времени обеспечивает целостный результат и способна функционировать в условиях распада на секции, но достигает этого в ущерб доступности: может не выдавать отклик на запись при падении одного из узлов.	**Нет**
5	**Строгий ACID**	Да	CAP консистентность требует линеарезуемости, но на практике нередко можно встретить даже Eventual Consistency вместо ACID в зависимости от настроек NoSQL СУБД.	**Нет**
6	**Постоянное дублирование актуальных данных (изменений) на всех живых узлах**	Да	Да	Нет, при отсечении может быть задержка дублирования.
7	**BASE – Basically Available, Soft-state, Eventually (Weak) Consistent** **противоположностью ACID-модели, утверждая, что истинная согласованность не может быть достигнута в реальном мире и смоделирована в высокомасштабируемых системах**	**Нет**	В зависимости от настроек?	Да, базовая доступность, неустойчивое состояние, согласованность в конечном счёте
8	**24/7 Availability (Доступность) A** **ВСЕГДА сохраняет корректный отклик (без возврата ошибок и таймаутов) ЛЮБОГО живого узла на ВСЕ запросы read (на практике)/write(в теории). На практике все же удается достичь A НЕ 100% времени, а только в лучшем случае в рамках SLA. Не подразумевает консистентность ответов в общем случае (например, для AP). Для достижения availability отсеченной секции достаточно корректного локального взаимодействия с клиентами.**	Все клиенты всегда могут читать и писать 24/7 Высокая латентность ответа не исключает высокую доступность.	Не гарантирует доступность. При распаде на секции может выдавать ошибку (например таймаут) вместо корректного ответа при попытке записи.	Все клиенты всегда могут читать и писать 24/7 Высокая латентность ответа не исключает высокую доступность.
9	**При падении одного из узлов**	Падение узла не влияет на работоспособность системы в целом (остальных живых узлов). Система продолжает работать в целом на остальных живых узлах, возможно увеличение latency ответов.	Продолжает корректно работать по крайней мере на чтение. Попытки записи будут обрываться или сильно задерживаться, пока система не убедится в своей консистентности.	Падение узла не влияет на работоспособность системы в целом (остальных живых узлов). Переживает падения части серверов, но когда они входят в строй, они будут выдавать пользователям старые неактуальные данные.
10	**Partition Tolerance P** **Сохраняет корректный отклик при развале сети, связующей узлы**	Система неустойчива при сетевых сбоях, которых однако в теории быть не должно, потому что сеть теоретически надежная. Но на практике такого к сожалению не бывает.	Способна функционировать в условиях распада, но достигает этого в ущерб доступности.	Способна функционировать в условиях распада, но достигает этого в ущерб консистентности.
11	**При разделении сети системы (Internal network failures, CAP теорема по сути знает только о таком виде сбоя)**	На практике в РСУБД обычно происходит задержка репликации ведомого отсеченного узла с репликой (потеря его консистентности). Останавливается запись на отсоединенный узел, на нем остаются возможны только операции чтения и то неконсистентные, если одновременно продолжать писать на мастер. Т.е. при отсечении пропадает доступность (на запись) и консистентность отсеченной ведомой части.	Расщепление (секционирование) распределённой системы на несколько изолированных (исчезает связь по сети между секциями системы) секций не приводит к некорректности отклика от каждой из секций. Противоречит C8 для CP?	
12		Разделения в CA системах быть не должно.	Проблемы доступности, система не может дать ответ на запрос клиента, вместо ошибки таймаута.	Проблемы консистентности. Система временно теряет актуальность данных при ответе на запрос клиента с отсеченных узлов.
13	**Вид используемой сети для связи узлов**	Максимально надежная локальная сеть (на практике все же возможны редкие сбои).	Асинхронная сеть с задержками, которая может терять или задерживать сообщения, что характерно для любой Интернет-системы. Такие сетевые взаимодействия допускают обрывы связи и потери пакетов.	
14	**Особенности СУБД**	Обычно РСУБД с репликацией для достижения CA требует очень надежной сети, которых на практике не бывает.	Обычно нереляционная СУБД NoSQL.	
15	**Примеры СУБД или другой распределенной системы, которые могут ПРИБЛИЗИТЕЛЬНО подходить под условия CAP пары при определенных настройках СУБД. Настройкой СУБД нередко можно менять ее класс по CAP и PACELC**	PostgreSQL, IBM Db2, ElasticSearch, LDAP, etc.	Apache Ignite, Redis, Memcache, HBase, Berkley DB, MongoDB, etc.	Cassandra, Riak, CoachDB, DynamoDB, SimpleDB, KAI, DNS, Web Caches etc.
16	**Взаимоотношения узлов, например Master-slave, etc.**	Master→ несколько readonly slaves. Commit может быть только синхронным (**EC**).	Несколько синхронных мастеров. Commit для клиента отрабатывает только после фактической записи на все ноды.	Несколько асинхронных мастеров для NoSQL. Commit для клиента отрабатывает раньше фактической записи на все ноды, т.е. реплики как минимум асинхронные (**EL**) или вообще отсеченные. Данные записываются на все узлы, но с задержкой, возможно после восстановления сетевой связности распределенной системы или как минимум с лагом для асинхронных реплик.
17	**Блокировки**	Commit может быть только синхронным (**EC**).	Пессимистичные для попытки обеспечения целостности	
18	**PACELC, где L – good (low) latency P – возможность наступление события разделения системы. If P then choose good A or C else good L or C**	**EC**	**PC**	**PA/EL**
19	**PA: If P then good A and bad C**			**PA**:PostgreSQL (async master-master), Cassandra, Riak MongoDB
20	**PC: If P then good C and bad A**		**PC**: MongoDB?	
21	**EC: else (if not P) then good C and bad (high) L**	**EC**:PostgreSQL (sync master-slave?), MongoDB		
22	**EL: else (if not P) then good (low) L and bad C**			**EL**:PostgreSQL (async master-slave), Cassandra, Riak
23				
24	**Ссылки на использованные материалы:**	Теорема CAP CAP Theorem Revisited Забудьте CAP теорему (как неактуальную) Всё, что вы не знали о CAP теореме Чем плоха CAP-теорема PACELC Теорема	ru.wikipedia.org/wiki/Теорема_CAP robertgreiner.com/cap-theorem-revisited/ habr.com/ru/post/258145/ habr.com/ru/post/328792/ bigdataschool.ru/blog/cap-alternatives-for-nosql-and-big-data.html en.wikipedia.org/wiki/PACELC_theorem	
25	**Фразы с форумов (не мои):** **Суть CAP:** Нельзя сделать быструю, надежную и дешевую систему одновременно. Хотите "**C**" - ждите всех соединений, и пусть все остальные отдыхают. Хотите "**A**" - не надейтесь на согласованность. **PACELC:** A high availability requirement implies that the system must replicate data. As soon as a distributed system replicates data, a trade-off between consistency and latency arises. **Serializability** гарантирует, что «существует» порядок последовательного выполнения транзакций, выполняемых параллельно. **Lineralizability** же также гарантирует, что этот порядок соответствует порядку инициации транзакций (раньше стартовала — раньше выполнена).			