2. NoSQL бази данни

I. Въведение

Терминът NoSQL е съкращение от „Not only SQL" и се използва за всички нерелационни бази данни. Този термин е въведен от Карло Строци през 1998 г. NoSQL базите данни са алтернатива на релационните бази данни. Те не ползват релационен модел на данните и не е задължително да поддържат SQL интерфейс. Основното им предназначение е да се справят с новите предизвикателства като: обработка в реално време на големи масиви от данни (Big Data), мащабируеми (основно хоризонтално) системи, системи с висока степен на отказоустойчивост и толерантни към грешки системи. Данните могат да се съхраняват не само в таблици. В случаите при които няма предварително зададени схеми за описание на данните (фиксиран брой колони) е за предпочитане да се използва нерелационна база данни. Типични примери за това са данните генерирани от GPS локацията на клиентите на дадена услуга; борсови данни; данни, генерирани от сензорни безжични мрежи от IoT инфраструктура; данни от социални мрежи и др. По принцип при всички изброени случаи се генерират много големи масиви от данни (милиони записи). Релационните бази данни трудно се справят с подобни обеми от данни тъй като трудно се мащабират хоризонтално – разпределени бази данни. Тук на помощ идват NoSQL базите данни. С тях можем лесно да избегнем сложните операции за обединяване на таблици, характерни за SQL. Много по-лесно се реализира и хоризонталното мащабиране.

Основните **предимства** на NoSQL базите данни са следните:

- Опростен дизайн.
- Не изискват прилагането на строга схема, която да важи за цялата база данни.
- По-лесна реализацията на хоризонтално мащабиране.
- Възможност за работа в реално време с много големи масиви от данни (Big Data), благодарение на ефективния начин за извличане и филтриране на данни чрез MapReduce.
- По-лесна реализация на силно разпределени бази данни.
- По-лесна реализация на синхронизация на данните между отделните сървъри за управление на базите данни при използване на силно разпределена архитектура.

Както релационните, така и NoSQL базите данни имат и **недостатъци**, например:

- Няма правила за стандартизация.
- Ограничени видове заявки (не поддържат стандартните SQL заявки).
- Не предлагат традиционни възможности на релационните бази данни, например съгласуваност.
- Когато обемът на данните се увеличава, е по-трудно да се поддържат уникални стойности като ключове.
- Не работят толкова добре със силно свързани данни.
- По-трудни за усвояване от начинаещи разработчици.
- Повечето са с отворен код, което ги прави от гледна точка на сигурност неатрактивни за бизнеса.

Основните разновидности на NoSQL базите данни са следните:

- **Документен** тип, например MongoDB, CouchDB и CouchBase.
- **Колонен** тип, например BigTable, Cassandra, HBase и Hypertable.
- **Key-value** тип, например Redis, Riak, Memcached и Scalaris.
- **Graph**-базирани, например Neo4j, HyperGraphDB, IngoGrid и Flock DB.

Документният тип бази данни свързват всеки ключ със структура от данни, която се нарича документ. Данните се съхраняват в документи, които могат да се обединяват в колекции. Документите могат да съдържат двойки ключ-масив или двойки ключ-стойност, както и вложени документи. Най-често документите се описват чрез JavaScript Object Notation (JSON) или Binary JSON (BSON) обекти. Примери за документни NoSQL са MongoDB, Apache CouchDB, Couchbase, IBM Domino, Cosmos DB и RavenDB..

При **ключ-стойност** базите данни всеки отделен елемент се съхранява като двойка ключ и стойности. Хранилищата за ключове и стойности са най-простата база данни сред всички NoSQL бази данни. Примери за Key-value NoSQL са Redis, Memcached, Apache Ignite, Riak и Scalaris.

Колонният тип бази данни са оптимизирани за заявки върху големи масиви от данни и вместо редове съхраняват заедно колони от данни. Примери за колонен тип NoSQL бази данни са HBase, Cassandra, Hypertable и Scylla.

Граф-базираните бази данни съхраняват информация чрез описание на връзките чрез графи. Пример за подобни връзки са връзки между потребителите на социални мрежи и транспортни връзки. Примери за граф-базираните бази данни са Neo4j, AllegroGraph, HyperGraphDB, IngoGrid и Flock DB.

II. ACID и BASE бази данни

2.1. ACID бази данни

При тези бази данни (основно релационни) се гарантират следните свойства: атомарност, консистентност, изолираност и издръжливост:

- Atomicy (атомарност) – всяко действие (транзакция) с базата данни или се изпълнява изцяло или не се изпълнява (връща се грешка). Ако една част от транзакция се провали, се анулира цялата транзакция.
- Consistency (консистентност, съгласуваност) – Всяка транзакция променя базата данни от едно състояние с консистентна информация в друго консистентно състояние.
- Isolation (изолация) – няма възможност за промяна на данни, които участват в все още незавършила операция. Това се гарантира с различни видове блокировки. Трябва да се има предвид, че тези блокировки има опасност да доведат до мъртва хватка и затова някои бази данни не гарантират 100% изолация.
- Durability (издръжливост) – записът на данни в базата гарантира бъдещата наличност на тези данни.

2.2 BASE бази данни

При тези бази данни (разпределени бази данни), с цел по-добра производителност, се гарантира консистентност само в края на всяка операция. Свойствата, които описват тези бази данни, са следните:

- Basically Available (базова наличност) – базата запазва функциите си, дори при блокиране на част от нодовете.

- Soft State (меко състояние) – това свойство е свързано със състоянието на данните в базата и означава: състоянието не е гарантирано консистентно; управлява се с вероятности; състоянието не може да бъде прекратено от потребител, а само от хардуера (сървър, мрежов хардуер); потребителят отговаря за състоянието на данните; състоянието на данните не е критично за услугата, която се предлага.
- Eventual consistency (евентуална консистентност) – консистентността не се гарантира на 100%, а само в края на всяка операция. Това означава, че ако в даден момент базата не се обновява, евентуално всички нодове на базата ще връщат актуалната и консистентна стойност на данните.

Съгласуваността (консистентността) бива няколко вида:

- Силна съгласуваност. След приключване на обновяването, всеки един клиент ще вижда обновената стойност на данните.
- Слаба съгласуваност. Не се гарантира, че следващите заявки, след заявката за обновяване. ще върнат обновената стойност, освен ако не са изпълнени предварително зададени условия. Периодът до изпълняване на тези условията е период на съгласуваност.
- Евентуална съгласуваност. Това е форма на слаба съгласуваност, при която системата гарантира, че ако няма направени нови обновявания на данните, евентуално всички нодове ще върнат вярната стойност.

Повечето NoSQL бази данни (CouchDB, MongoDB, Cassandra) гарантират евентуална консистентност. Например, **CouchDB** гарантира евентуална консистентност чрез последователни репликации. За да е възможно се задава версия на всеки документ, а старите документи не се изтриват. При **Cassandra** е възможно да се зададе каква консистентност искаме да получим още при генериране на заявката към базата данни. При четене на данни, стойностите за ‘consistency level’ могат да бъдат: ONE (връща желаната информация от първия достъпен нод), QUORUM (стойността трябва да е обновена в 50%+1 от нодовете), ALL (всички нодове са обновили желаните данни). При запис на данни, стойностите за ‘consistency level’ могат да бъдат: ZERO (без налагане на блокировки), ONE (изчаква се поне един нод да потвърди записа), QUORUM (изчаква се 50%+1 от нодовете да потвърдят записа), ALL (изчаква се всички нодове да потвърдят записа). При стойност ALL на ‘consistency level’ Cassandra гарантира консистентност на данните. **MongoDB** поддържа няколко вида евентуална консистентност:

- Single Writer Eventual Consistency. Евентуална консистентност с един източник на промени. Клиентът може да прочете остаряла информация или да получи изменения в неправилен ред.
- Monotonic Read Consistency. Евентуална консистентност, в която не може да се получат изменения в неправилен ред.
- Read-your-own-writes Consistency. Евентуална консистентност, при която клиентът достъпва собствените си обновявания.

Теорема CAP (Consistency, Availability, Partition Tolerance)

За нерелационните бази данни е в сила теоремата на Ерик Брюър, известна като Consistency, Availability, Partition tolerance (CAP) теорема. Тя гласи, че за една разпределена база данни не може да се гарантира едновременно всички данни да са

налични, да са разделени на части между различните сървъри и структурата между тях да се съхрани.

Consistency (съгласуваност): Всеки клиент, независимо чрез кой нод се обслужва, получава едни и същи данни както останалите клиенти, независимо от протичащите конкурентните обновления (всички реплики на документите имат една и съща версия).

Availability (наличност): Дори да има нефункциониращи нодове, клиентите имат възможност да четат и записват информация.

Partition tolerance (възможност за разделяне на части): Базата данни може да бъде разпределена и да се обслужва от множество сървъри. Системата остава работоспособна при наличие на проблеми при някой от сървърите.

Всяка база данни в един момент от време гарантира два от тези три параметри от CAP теоремата:

- **AP** бази данни: CouchDB, Cassandra, SimpleDB, Riak, Dynamo.
- **CP** бази данни: Couchbase, MongoDB, Big Table, Hypertable, Hbase, Redis.
- **CA** бази данни: Всички релационни бази данни, например MySQL.

Когато информацията в базата данни е над определен критичен обем, става невъзможно достъпването ѝ в реално време, ако тази информация е съсредоточена в една база данни (един хост). В този случай се налага изграждане на клъстер от сървъри, които обслужват заявките на клиентите. Такива бази данни е прието да се наричат **разпределени бази данни**. При тях се намалява значително времето за обслужване на заявките на клиентите, но на преден план остава друг проблем – гарантиране на съгласуваност на информацията. Ако един клиент обнови информация в базата данни на един от сървърите, клиентите, които се обслужват от другите сървъри и адресират същата информация, трябва да получат последната нейна стойност. За да е възможно това се налага поддържането на синхронизация на информацията в отделните бази данни. Най-лесният, но и най-бавен начин за гарантиране на съгласуваност е клиентите да изчакат докато информацията от която се интересуват се обновява. Този начин обаче не гарантира достъпност до информацията за определени интервали от време.

Ако достъпността е приоритет пред съгласуваността системата трябва да предостави възможност за запис на информация в някой от нодовете на системата и след това да има механизъм със забавяне във времето да се гарантира и съгласуваност на информацията. В този случай се казва, че базата поддържа евентуална съгласуваност. Такава база данни например е CouchDB. Вместо да се блокира достъпа до базата по време на обновяване на информация, CouchDB използва Multi-Version Concurrency Control (MVCC), за да се обслужат конкурентните заявки до базата данни. Идеята е за всеки документ да се поддържа версия, която се отразява автоматично чрез стойността на свойство с име „_rev”. Обновяването на съдържанието на документ води автоматично до създаване на нов документ, който съдържа най-новата информация, но старият документ не се изтрива. Тази стратегия позволява обслужване на конкурентните заявки без значимо забавяне на отговорите. Нека клиент A чете данни от документ X. В същият момент клиент B прави опит за запис в документ X. Записът започва веднага, без да се чака транзакцията с клиент A да завърши, тъй като за клиент B се създава нов документ с нова версия. Ако клиент C желае да прочете данни от същия документ, той автоматично ще бъде насочен да прочете съдържанието на най-новата версия на документа.

III. Класация на базите данни

В Табл. 1 е показана класацията за всички бази данни за периода юни 2020 – юни 2021. Информацията е от сайта https://db-engines.com/en/ranking. Вижда се, че в Топ 10 на този етап попадат само две NoSQL бази данни – MongoDB и Redis. Причината за малкият брой NoSQL бази данни в класацията Топ 10 е основно огромният брой вече инсталирани релационни бази данни, които трябва да се поддържат. Дори да е необходимо преминаване към NoSQL база данни процесът на миграция е много бавен и скъп.

Табл. 1 Класация Топ 10 на всички бази данни за периода юни 2020 – юни 2021

Rank			DBMS	Database Model	Score		
Jun 2021	May 2021	Jun 2020			Jun 2021	May 2021	Jun 2020
1.	1.	1.	Oracle	Relational, Multi-model	1270.94	+1.00	-72.65
2.	2.	2.	MySQL	Relational, Multi-model	1227.86	-8.52	-50.03
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	991.07	-1.59	-76.24
4.	4.	4.	PostgreSQL	Relational, Multi-model	568.51	+9.26	+45.53
5.	5.	5.	MongoDB	Document, Multi-model	488.22	+7.20	+51.14
6.	6.	6.	IBM Db2	Relational, Multi-model	167.03	+0.37	+5.23
7.	7.	↑ 8.	Redis	Key-value, Multi-model	165.25	+3.08	+19.61
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	154.71	-0.65	+5.02
9.	9.	9.	SQLite	Relational	130.54	+3.84	+5.72
10.	10.	↑ 11.	MS Access	Relational	114.94	-0.46	-2.24

В Табл. 2 е показана същата класация, но само за NoSQL базите данни. Освен MongoDB и Redis в Топ 10 за NoSQL базите данни са Cassandra, Amazon DynamoDB и Neo4j.

Табл. 2. Класация за NoSQL базите данни за периода юни 2020 – юни 2021

Rank			DBMS	Database Model	Score		
Jun 2021	May 2021	Jun 2020			Jun 2021	May 2021	Jun 2020
5.	5.	5.	MongoDB	Document, Multi-model	488.22	+7.20	+51.14
7.	7.	↑ 8.	Redis	Key-value, Multi-model	165.25	+3.08	+19.61
11.	11.	↓ 10.	Cassandra	Wide column	114.11	+3.18	-4.90
16.	16.	16.	Amazon DynamoDB	Multi-model	73.76	+3.69	+8.90
18.	↑ 19.	↑ 22.	Neo4j	Graph	55.75	+3.52	+7.48
23.	23.	↓ 21.	HBase	Wide column	43.52	+0.27	-5.22
28.	↓ 26.	↓ 25.	Couchbase	Document, Multi-model	29.07	-1.16	-0.07
30.	30.	↓ 27.	Memcached	Key-value	25.18	+0.68	+0.37
39.	39.	↓ 35.	CouchDB	Document, Multi-model	16.12	+0.15	+0.05

81.	↑ 83.	↓ 77.	Oracle NoSQL	Multi-model	4.31	+0.61	+0.09
90.	↓ 87.	↓ 81.	RavenDB	Document, Multi-model	3.50	+0.25	-0.32
91.	↓ 89.	↓ 82.	IBM Cloudant	Document	3.36	+0.25	-0.32

NoSQL базите данни са съвременна технология, която намира все по-широко приложение, тъй като добре се интегрира с идеите за BigData и IoT. Пионерите в теза технология са Google със своята BigTable и Amazon със Dynamo. На базата на идеите, заложени в BigTable и Dynamo, се създава Cassandra, която се използва Facebook. От архитектурата на Google BigTable е инспирирана разработката на Hadoop, а от архитектурата на Dynamo - Riak.

В Табл. 3 са описани основните параметри на най-често използваните NoSQL бази данни.

Табл. 3. Параметри на често използваните NoSQL бази данни

Параметър	**База данни**					
	CouchDB	**Couchbase**	**MongoDB**	**Cassandra**	**Neo4j**	**Redis**
Тип	документен	документен	документен	колонен	граф	Key-value
Език	Erlang	Erlang, C	C++	Java	Java	C
Заявки	HTTP/REST (JSON)	N1QL (JSON, BSON)	HTTP/REST (JSON, BSON)	CQL3	HTTP/REST (XML, Protobuf, binary)	Lua script
Особености	Master-master репликации, MVCC (запис операциите не блокират четене операциите) Map-reduce Doc validation	Master-master и cross-datacenter репликации, clustering, moxi, Много бърза, Memcache съвмвестим	Master-slave репликации Shardind Doc валидиране Aggregation framework, Geospatial indexing	Липсват join и aggregation, Key-базирани заявки, Map-reduce Hadoop, cross-datacenter репликации, Java triggers	Map-reduce Hadoop, real-time query optimization	Master-slave репликации, key-базирани структури данни, bit и bitfield операции, Geo API, expired time support
Подходяща за	Съхранение на рядко променящи се данни	Low latency data access, High concurrency and availability	Динамично променящи се заявки	Анализ на Web данни, данни от сензорни масиви	Best map-reduce engine	Динамично променящи се данни
Приложение	CMS, CRM, Multi-site deployment	Big Data Real-time apps	Big Data	Big Data	Search engines	Real-time системи

IV. Проектиране на нерелационни бази данни

Проектирането на нерелационни бази данни се реализира основно на базата на шаблони при които се отчитат добрите практики в областта на проектиране на всяка конкретна база данни.

При проектиране на базата данни трябва да се следи за:

- Намаляване на излишните данни в базата чрез разделяне на изходната информация на необходимия брой таблици.
- Правилен подбор на първичните и външни ключове.
- Отговаря ли базата данни на изискванията на клиента от гледна точка на надеждност, сигурност, бързодействие, мащабируемост и изготвяне на справки и отчети?

Релационните бази данни имат фиксирана **схема** на описание на данните. Полетата от таблиците имат фиксиран тип и дори да не се налага да приемат в даден момент стойност, те трябва да имат стойност, дори тя да е null. Това води до неефективно използване на паметта. За нерелационните бази данни често се казва, че са бази без схема. На практика всяка база данни има схема, но при NoSQL базите данни схемата е *динамична*. Това означава, че не е наложително схемата твърдо да се прилага за всички документи в дадена колекция. Напълно допустимо е да липсват определени полета в част от документите. Схемата се дефинира на ниво приложение. Тя трябва да може да се развива с промяната на приложението. Такава схема се нарича и *полиморфна*.

Дизайнът на схемата на базата данни е важен процес тъй като от него до голяма степен зависи производителността на цялата услуга, както и мащабируемостта на базата данни. Преди да проектирате базата данни, трябва да отговорите на редица въпроси, например:

1. Кои са обектите с които приложението ви работи и които трябва да са част от база данни?
2. Колко и от какъв тип са връзките между тези обекти (1:1, 1:N или M:N)?
3. Колко е броят на потребителите на вашето приложение и колко често ще се извлича информация от базата данни?
4. Колко често се налага вмъкване на нови обекти в базата данни?
5. Колко често се налага изтриване на обекти от базата данни?
6. Колко често се налага модифициране на съдържанието на обекти от базата данни?
7. Какви заявки до базата данни ще реализира приложението ви?
8. Как ще се реализира достъпа до обектите от базата данни – чрез единичен ключ, чрез съставен ключ, чрез стойността на свойства или чрез прилагане на някакъв вид филтриране и сортиране на данните?

Планиране на базата данни

Преди да проектирате базата данни, трябва да отговорите на редица въпроси, например:

9. Кои са обектите с които приложението ви работи и които трябва да са част от база данни?
10. Колко и от какъв тип са връзките между тези обекти (1:1, 1:N или M:N)?
11. Колко е броят на потребителите на вашето приложение и колко често ще се извлича информация от базата данни?
12. Колко често се налага вмъкване на нови обекти в базата данни?

13. Колко често се налага изтриване на обекти от базата данни?
14. Колко често се налага модифициране на съдържанието на обекти от базата данни?
15. Как ще се реализира достъпа до обектите от базата данни – чрез единичен ключ, чрез композитен ключ, чрез стойността на свойства или чрез прилагане на някакъв вид филтриране на данните?

Нормализация на базата данни

След като отговорите на всички по-горе споменати въпроси, ще можете да изберете подходящата за вашето приложение тип база данни. Независимо от нейния тип проектирането й трябва да завърши с нормализация на данните. Нормализацията е процес, който има за задача да намали излишъка и взаимозависимостта на информацията в базата. За целта трябва да се прецени кои обекти да се опишат в отделен документ и кои документи да се обединят в колекции. Едно от предимствата на нормализацията е, че базата ще стане по-малка по размер (няма да има повторение на обекти в рамките на една колекция. Един от недостатъците на нормализацията е, че се увеличават броя на заявките – получаването на крайният резултат може да изисква заявки към множество взаимосвързани документи. Когато връзката между два обекта е 1:N, то най-логично е двата обекта да се моделират в два различни документа. Например, потребителите на една книжарница и книгите, които те заемат трябва да са отделни документи: всеки потребител може да заема множество книги и няколко потребители могат да заемат едни и същи книги. Книгите, които всеки потребител заема, ще се кодира с id на книгата, а за всяка книга ще има отделен документ в който се описва както id, така и автора, наименованието, броя страници, жанра на книгата и други.

Денормализация на базата данни

Денормализацията на базата е процес при който се търсят обекти с връзки от тип 1:1. В този случай тези обекти трябва да се моделират чрез един документ. Това обаче трябва да се направи само ако обектите не са големи по размер и не се обновяват често. Предимството на денормализацията е бързодействието на обслужване на заявките – необходимо е да се достъпи само един документ, който съдържа цялата необходима информация. Например, клиентите на дадена услуга могат да имат два адреса: домашен и на работното си място. Тук въпросът е дали тези адреси да са част от документа, който описва клиент или да бъдат отделни документи. Тъй като домашният адрес е сравнително уникален за всеки клиент е нормално той да е част от документа, който описва клиента. Множество клиенти обаче може да имат един и същ работен адрес. Ако се интересувате често кои от вашите клиенти работят на едно и също място, то трябва да интегрирате работния адрес в документа, който описва клиента. В противен случай работното място трябва да е отделен документ, за да се намали размера на документите, описващи клиентите.

Нормализация vs денормализация

Заявките към една база данни могат да бъдат Create, Read, Update и Delete (CRUD). Критични са операциите при които се реализира промяна на съдържанието на базата данни. Целта е всички клиенти на услугата да получат последно модифицираните данни, дори когато базата е с разпределена архитектура. Следователно, операциите, свързани със запис, трябва да са atomic. Това означава, че само един процес трябва да има права да обновява един документ или колекция в даден момент от време. Ако документа е

денормализиран, атомичността (невъзможни са блокировки на данни) на операция запис се гарантира лесно. Това не е в сила за нормализираните документи. В този случай едно обновяване може да изисква запис в множество документи. Повечето NoSQL бази данни решават проблема с атомичността на операции запис чрез репликиране на базата данни (дублициране на данните на множество инстанции на базата, които формират клъстер).

Избор на правилния брой на документите

Основателен е въпросът дали да се работи с много на брой документи, или с малко документи, които съдържат много на брой свойства и техните стойности. Ако бързодействието е важният за вас параметър, то трябва да се работи с колкото се може по-малко на брой документи, тъй като дисковите операции са много бавни в сравнение с операциите в паметта. Ако даден документ в даден момент не е необходим за функционирането на вашата система, то най-добре е той да бъде изтрит. Затова още при проектирането трябва да се дефинира времето на живот на всеки документ.

Предимства на NoSQL базите данни:

- Възможност за обработка на големи обеми данни
- Няма единична точка на отказ
- Лесно репликиране
- Няма нужда от отделен слой за кеширане
- Осигурява бърза производителност и хоризонтална мащабируемост.
- Може да обработва структурирани, полуструктурирани и неструктурирани данни с еднакъв ефект
- Използва се Обектно-ориентирано програмиране
- NoSQL базите данни не се нуждаят от специален високопроизводителен сървър
- Поддръжка на множество базови езици и платформи за разработчици
- Лесни за внедряване в сравнение с използването на RDBMS
- Може да служи като основен източник на данни за онлайн приложения
- Работят с големи обеми данни
- Позволяват създаване разпределени бази данни и работа с множество центрове за данни
- Премахва необходимостта от специален слой за кеширане на данни
- Предлага гъвкав дизайн на схемата, който може лесно да се променя, без да се налага прекъсване на работа

Недостатъци на NoSQL базите данни:

- Няма правила за стандартизация
- Ограничени видове заявки
- Не предлагат традиционни възможности на релационните бази данни, например съгласуваност, когато няколко транзакции се извършват едновременно.
- Когато обемът на данните се увеличава, е трудно да се поддържат уникални стойности като ключове
- Не работи толкова добре с релационни данни
- Трудни за усвояване от начинаещи разработчици
- Повечето са с отворен код, което ги прави неатрактивни за бизнеса.

Основни изводи:

- NoSQL са нерелационни бази данни, която не изисква фиксирана схема, избягва JOIN операциите и са лесни за мащабиране
- Концепцията за NoSQL бази данни стана популярна с развитието на Интернет гиганти като Google, Facebook, Amazon и др., които работят с огромни обеми данни
- През 1998 г. Карло Строци използва термина NoSQL за своята олекотена база данни с отворен код
- NoSQL бази данни никога не следват релационния модел, те са или без схеми, или имат облекчени схеми
- Основните видове NoSQL бази данни са следните: 1) Базирани на двойки ключ-стойност; 2) Колонен тип бази данни; 3) Базирани на графи и 4) Документно-ориентирани бази данни (JSON, BSON).
- NoSQL базите данни може да обработват структурирани, полуструктурирани и неструктурирани данни с еднаква ефективност
- Теоремата CAP се състои от три думи Consistency (последователност), Availability (наличност) и Partition Tolerance (толерантност към разделяне)
- BASE е съкращение от Basically Available (основно наличен), Soft State (меко състояние), Eventual consistency (евентуална съгласуваност)
- Терминът "евентуална съгласуваност" означава да имате копия на данните на няколко машини, за да да се постигне висока наличност и мащабируемост
- NOSQL базите данни предлагат ограничени възможности за заявки при сравнение с традиционните релационни данни, които използват SQL заявки.