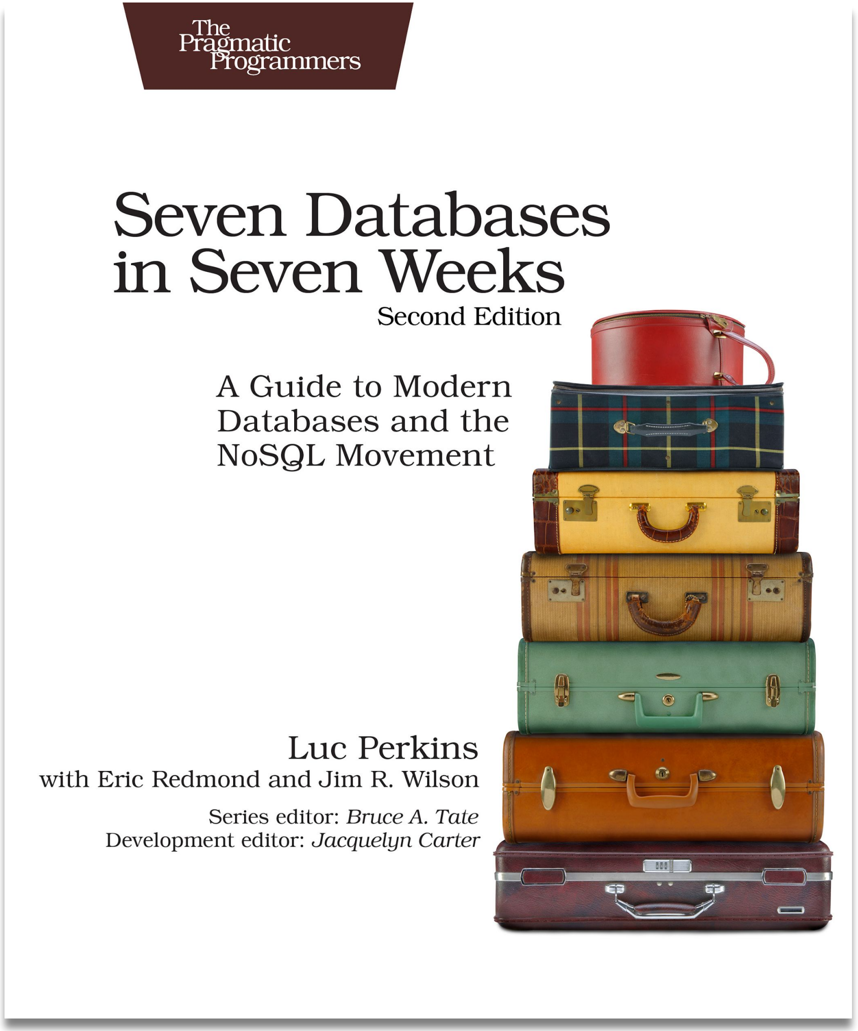
base de datos 2



The Pragmatic Programmers

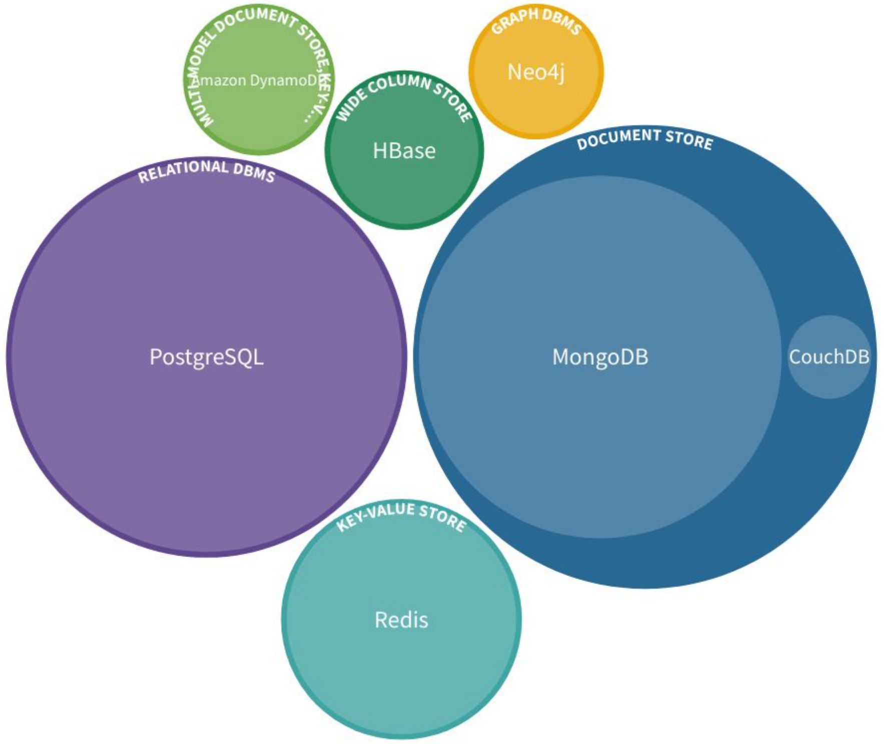
Seven Databases in Seven Weeks

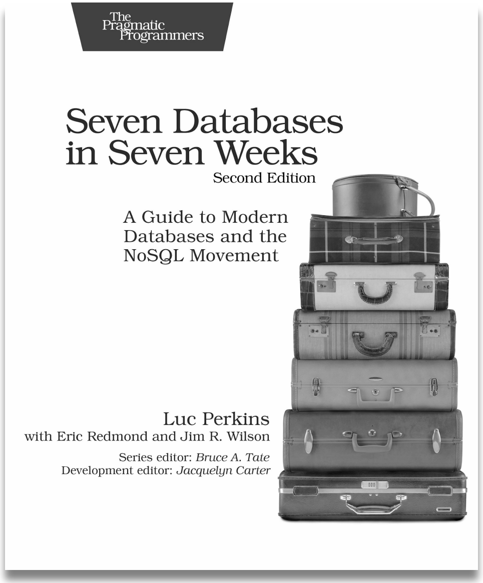
Second Edition

A Guide to Modern Databases and the NoSQL Movement

Luc Perkins
with Eric Redmond and Jim R. Wilson

Series editor: *Bruce A. Tate*
Development editor: *Jacquelyn Carter*




The Pragmatic Programmers

Seven Databases in Seven Weeks

Second Edition

A Guide to Modern Databases and the NoSQL Movement

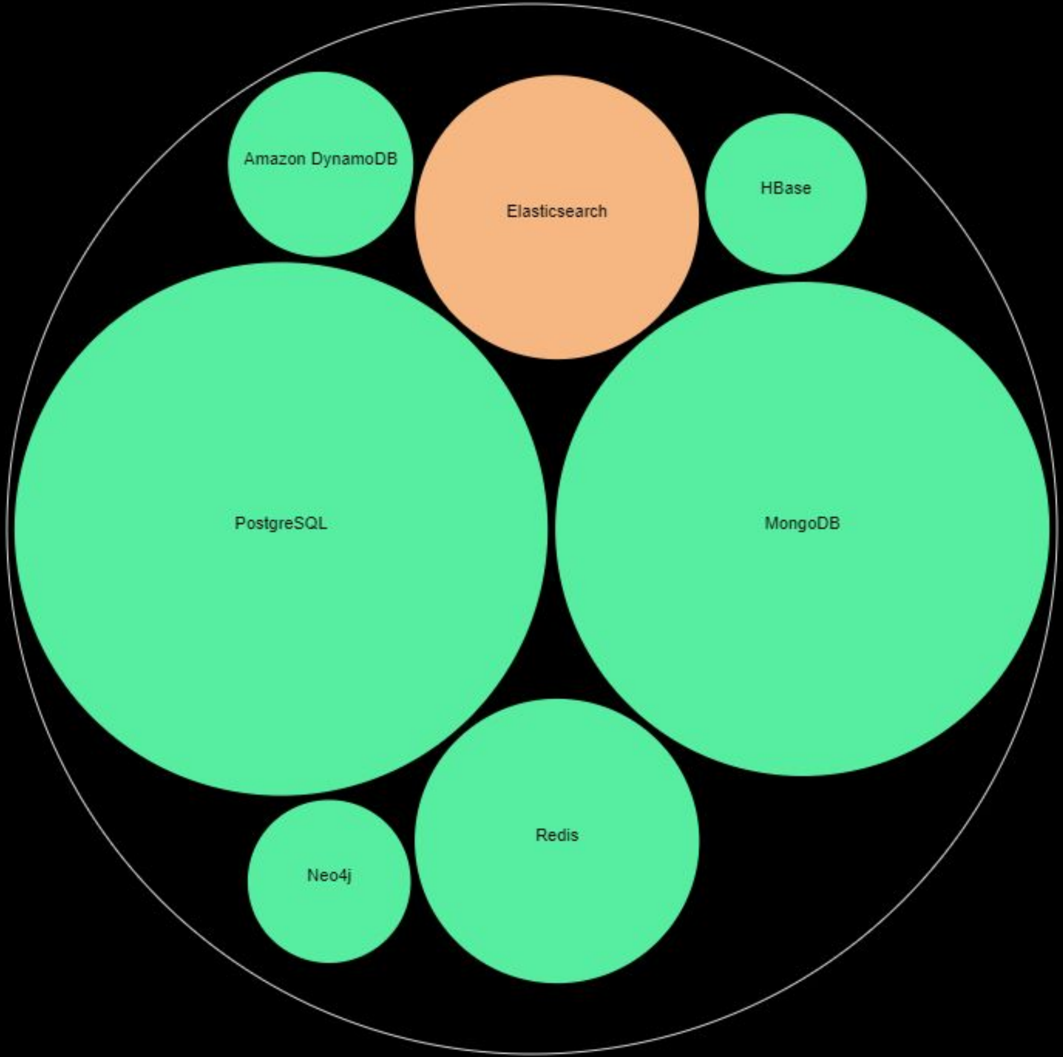
Luc Perkins

with Eric Redmond and Jim R. Wilson

Series editor: *Bruce A. Tate*

Development editor: *Jacquelyn Carter*

http://bit.ly/itba7dbs


Amazon DynamoDB
Elasticsearch
HBase
PostgreSQL
MongoDB
Neo4j
Redis

Database	Paradigm
PostgreSQL	Relational
HBase	Columnar
MongoDB	Document store
CouchDB	Document store
Neo4j	Graph database
DynamoDB	Cloud key/value store
Redis	Key/value

Polyglot persistence

La ***persistencia políglota*** es el concepto de utilizar diferentes tecnologías para manejar diferentes necesidades de almacenamiento de datos dentro de una aplicación de software determinada. La programación políglota, un término acuñado por Neal Ford en el 2006, expresa la idea de que las aplicaciones informáticas deben escribirse en una combinación de diferentes lenguajes de programación, a fin de aprovechar el hecho de que los diferentes lenguajes son adecuados para abordar diferentes problemas.

Este mismo concepto lo podemos aplicar a las *bases de datos*, ya que una aplicación puede comunicarse con diferentes bases de datos, utilizando a cada base de datos en lo que mejor se desempeña.

Comentarios sobre el TP Final/Especial

1. Encontrar un problema donde aplicar persistencia políglota.
    a. Justificar el uso
    b. Desarrollar una propuesta
    c. Implementar (para el Final)

~~2. Benchmark entre bases de datos~~
    ~~a. Seleccionar un conjunto de datos "grande"~~
    ~~b. Establecer los casos de prueba~~
    ~~c. Tomar los tiempos~~
    ~~d. Analizar los resultados~~

~~3.~~ Modelar e implementar un prototipo de un sitio X (por ej. **bit.ly**, **Notion**, ...) utilizando persistencia políglota.

ADC (1981)
ACID (1983)
CAP (2000)
BASE (2008)

1981 Jim Gray (ADC)

The Transaction Concept: Virtues and Limitations

El concepto de transacción surge con las siguientes propiedades:

Consistencia: la transacción debe cumplir con los protocolos legales.

Atomicidad: o sucede o no sucede; o todos están obligados por el contrato o ninguno está. Todo o nada.

Durabilidad: una vez que se confirma una transacción, no se puede derogar.

1983, Andreas Reuter & Theo Härder (ACID)

Principles of transaction-oriented database recovery

Coined the acronym ACID as shorthand for Atomicity, Consistency, Isolation, and Durability.

These four properties, atomicity, consistency, isolation, and durability (ACID), describe the major highlights of the transactional paradigm, which has influenced many aspects of development in database systems.

ACID (Atomicity, Consistency, Isolation, Durability)

Atomicity

Transactions are often composed of multiple statements. Atomicity guarantees that each transaction is treated as a single "unit", which either succeeds completely, or fails completely: if any of the statements constituting a transaction fails to complete, the entire transaction fails and the database is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors and crashes.

ACID (Atomicity, Consistency, Isolation, Durability)

Consistency

Consistency ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants: any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof. This prevents database corruption by an illegal transaction, but does not guarantee that a transaction is correct.

ACID (Atomicity, Consistency, Isolation, Durability)

Isolation

Transactions are often executed concurrently (e.g., reading and writing to multiple tables at the same time). Isolation ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially. Isolation is the main goal of concurrency control; depending on the method used, the effects of an incomplete transaction might not even be visible to other transactions.

ACID (Atomicity, Consistency, Isolation, Durability)

Durability

Durability guarantees that once a transaction has been committed, it will remain committed even in the case of a system failure (e.g., power outage or crash). This usually means that completed transactions (or their effects) are recorded in non-volatile memory.

Eric Brewer, CAP (2000)

El teorema CAP, también llamado conjetura de Brewer, enuncia que es imposible para un sistema de cómputo *distribuido* garantizar simultáneamente:

1. La consistencia (*Consistency*), es decir, que todos los nodos vean la misma información al mismo tiempo.

2. La disponibilidad (*Availability*), es decir, la garantía de que cada petición a un nodo reciba una confirmación de si ha sido o no resuelta satisfactoriamente.

3. La tolerancia a las particiones (*Partition Tolerance*), es decir, el sistema sigue funcionando incluso si algunos nodos fallan.

CAP

Consistency: A read is guaranteed to return the most recent write for a given client.

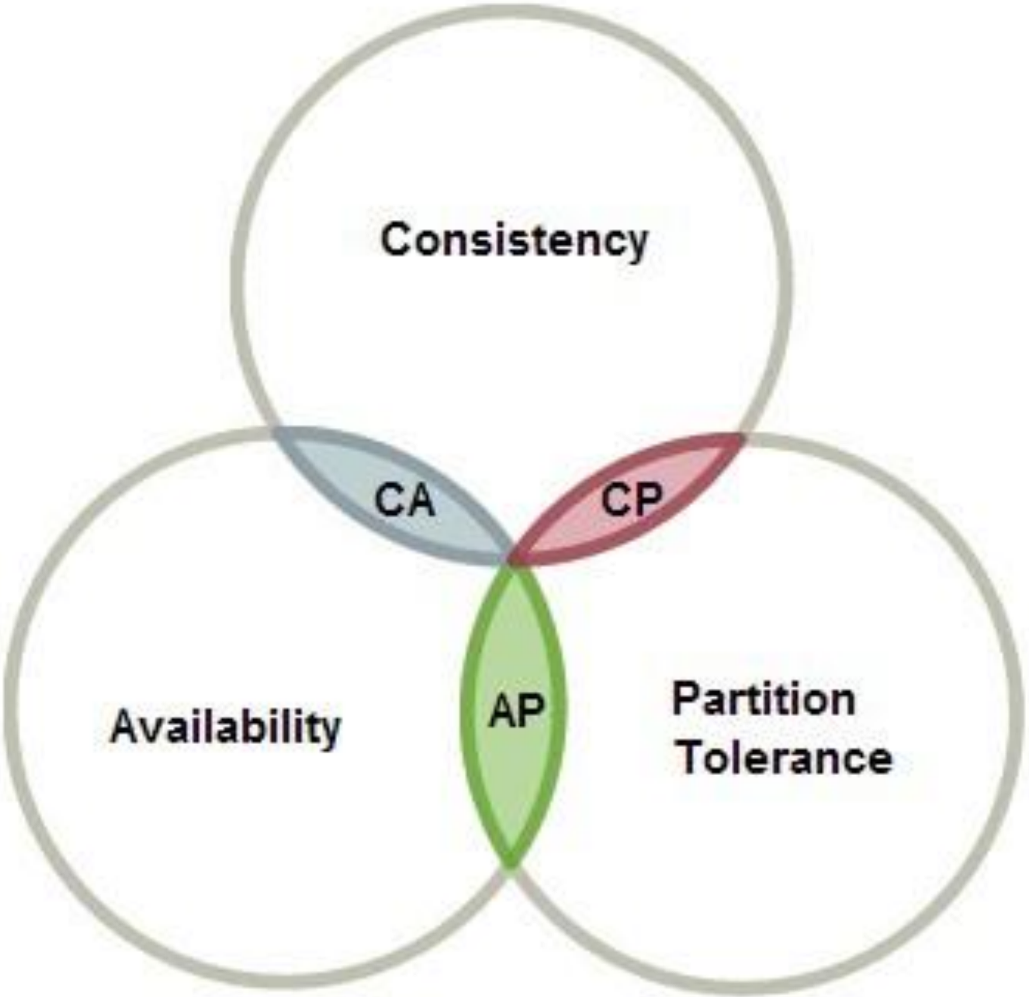
Availability: A non-failing node will return a reasonable response within a reasonable amount of time (no error or timeout).

Partition Tolerance: The system will continue to function when network partitions occur.

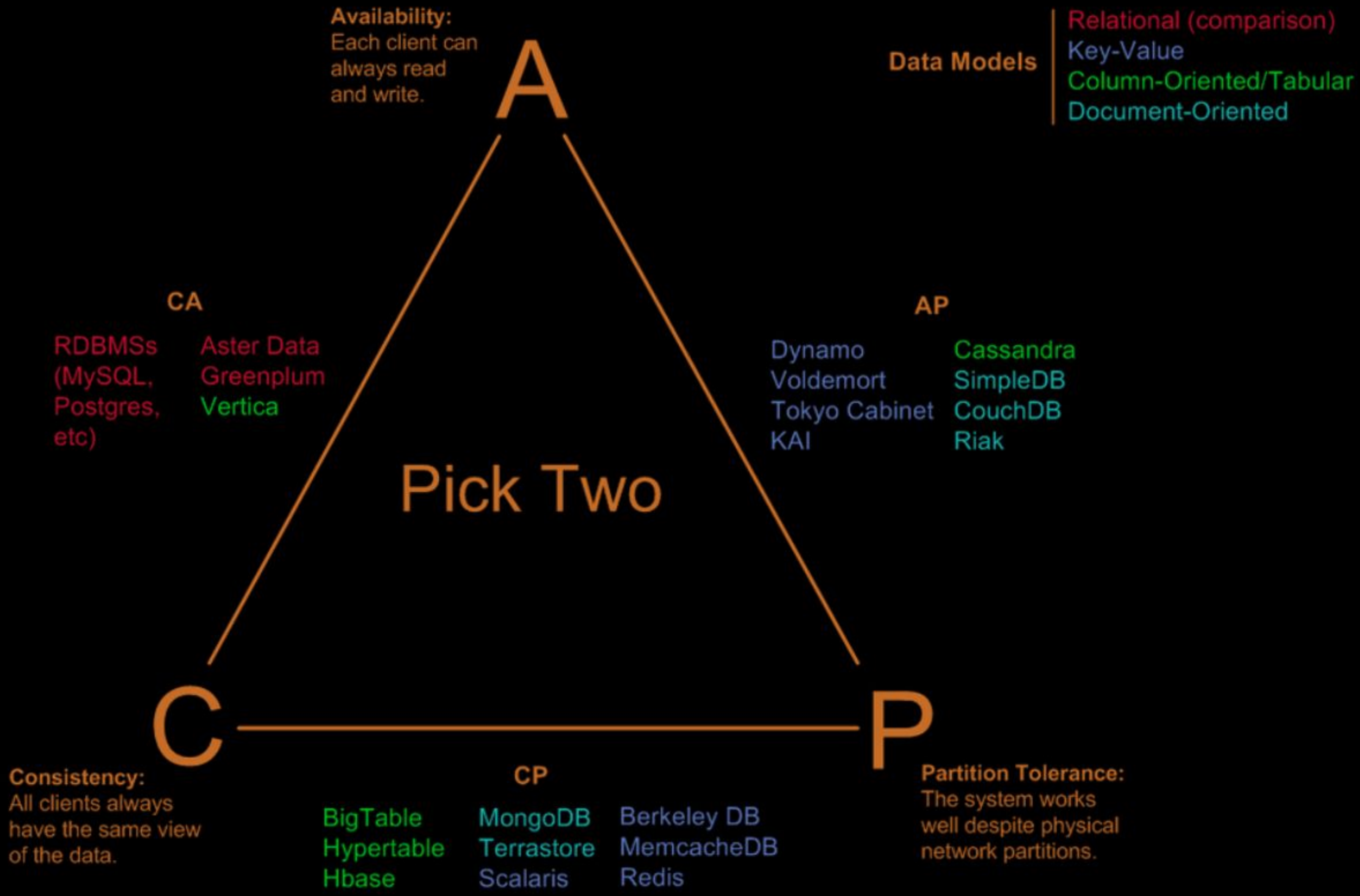
Consistency: Every read receives the most recent write or an error.

Availability: Every request receives a (non-error) response – without guarantee that it contains the most recent write.

Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.


Consistency
CA
CP
AP
Availability
Partition Tolerance

Visual Guide to NoSQL Systems



You can’t sacrifice Partition Tolerance

*Of the CAP theorem Consistency, Availability, and Partition Tolerance, Partition Tolerance is mandatory in distributed systems. You cannot not choose it. Instead of CAP, you should think about your availability in terms of **yield** (percent of requests answered successfully) and **harvest** (percent of required data actually included in the responses) and which of these two your system will sacrifice when failures happen.*

2010 https://codahale.com/you-cant-sacrifice-partition-tolerance/

2012, Eric Brewer, University of California, Berkeley

CAP Twelve Years Later: How the “Rules” Have Changed →
https://www.computer.org/cms/Computer.org/ComputingNow/homepage/2012/0512/T_CO2_CAP12YearsLater.pdf

2015 “A Critique of the CAP Theorem”, de Martin Kleppmann →
https://arxiv.org/pdf/1509.05393.pdf

Please stop calling databases CP or AP →
https://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html

más información sobre el el teorema CAP

- https://en.wikipedia.org/wiki/CAP_theorem
- Brewer, E. (2000). Towards Robust Distributed System. Symposium on Principles of Distributed Computing (PODC).
- Brewer, E. (2012, February). CAP twelve years later: How the "rules" have changed. Computer, vol. 45, no. 2, pp. 23-29.
- 2014 - Google Cloud Platform Live: Fireside Chat with Urs Hölzle, Jeff Dean, and Eric Brewer

Discutir casos, en los cuales nos interesa que ocurra:

1. CA (Consistency & Availability)
2. CP (Consistency & Partition Tolerance)
3. AP (Availability & Partition Tolerance)

BASE (2008)

An ACID Alternative, propuesto por **Dan Pritchett**, EBAY.

Eventually-consistent services are often classified as providing **BASE** (**B**asically **A**vailable, **S**oft state, **E**ventual consistency) semantics.   BASE ≈ Eventual consistency

Eventual consistency is a consistency model used in *distributed computing* to achieve *high availability* that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.

BASE (2008)

(B)asically (A)vailable: basic reading and writing operations are available as much as possible (using all nodes of a database cluster), but without any kind of consistency guarantees (the write may not persist after conflicts are reconciled, the read may not get the latest write)

(S)oft state: without consistency guarantees, after some amount of time, we only have some probability of knowing the state, since it may not yet have converged

(E)ventually consistent: If the system is functioning and we wait long enough after any given set of inputs, we will eventually be able to know what the state of the database is, and so any further reads will be consistent with our expectations

BASE (2008)

Básicamente (**A**) disponible: las operaciones básicas de lectura y escritura están disponibles tanto como sea posible (utilizando todos los nodos de un grupo de bases de datos), pero sin ningún tipo de garantías de consistencia (la escritura puede no persistir después de que los conflictos se reconcilian, la lectura puede no obtener la última escritura)

Soft a menudo: sin garantías de coherencia, después de cierto tiempo, solo tenemos alguna probabilidad de conocer el estado, ya que es posible que aún no haya convergido

Eventualmente consistente: si el sistema está funcionando y esperamos lo suficiente después de cualquier conjunto de entradas, eventualmente podremos saber cuál es el estado de la base de datos, por lo que cualquier lectura adicional será consistente con nuestras expectativas.

	Consistency concepts	**Transaction concepts**
Postgresql	Immediate Consistency	ACID
HBase	Immediate Consistency, Eventual Consistency	Single row ACID (across millions of columns)
MongoDB	Eventual Consistency, Immediate Consistency	Multi-document ACID Transactions with snapshot isolation
Redis	Eventual Consistency, Strong eventual consistency with CRDTs	Optimistic locking, atomic execution of commands blocks and scripts
DynamoDB	Eventual Consistency, Immediate Consistency	ACID
Neo4J	Causal and Eventual Consistency configurable in Causal Cluster setup, Immediate Consistency in stand-alone mode	ACID
CouchDB	Eventual Consistency	no