About This Article

NoSQL is a word that mystify many software engineers, In this article, I fully explain what the NoSQL is about and who cares.

Omid
10/19/2016

NoSQL

Omid Askary Golestany

شاید کلمه NoSQL کمی شما را بترساند، یا اصلا توجه شما را جلب نکند. در این مقاله قصد دارم شما را با NoSQL آشنا کنم و کمی از اصول و مفاهیم مهمی که در آن بکار می‌رود را معرفی کنم. توجه کنید با اینکه من از منابع معتبری در این نوشتار بکار بردم، این مقاله نمی‌تواند یک منبع کامل برای استدلال‌ها و مقایسه‌هایی که در این نوشتار انجام می‌شود، باشد. همچین پیشرفت این دانش به سرعتی هست که شاید در زمان مطالعه این نوشتار، خیلی از اطلاعات، دیگر صحت نداشته باشد. به همین علت این نوشتار نباید پایان مطالعه شما درباره NoSQL باشد.

توجه کنید که پیشنیاز درک و مطالعه این نوشتار، آشنایی با مفاهیم و اصول RDBMSها می باشد. اگر با این مفاهیم آشنا نیستید، پیشنهاد می‌کنم قبل از مطالعه، مفاهیم آن را بررسی و مطالعه کنید.

درنهایت از تمامی دوستانی که در نوشتن و فراهم سازی این مقاله به من کمک کردند، صمیمانه تشکر می‌کنم. درصورت وجود هرگونه پرسش، بخصوص انتخاب یک Object Database می توانید با من تماس بگیرید.

امید عسکری گلستانی

P.N: 09375349055

Email: hopegolestany@gmail.com

NoSQL چیست؟

شاید بتوان گفت تعریف کردن NoSQL ، از سختترین کارهایی هست که در زمان نوشتن این مقاله میتوان انجام داد. زمانی که من میخواهم به شخصی دیگر بگویم که کلاً دیتابیس چیست، معمولاً از مفاهیم RDBMS ها استفاده میکنم. علت این هست که RDBMS ها بسیار پخته هستند و مفاهیم آن استاندارد شده است و همین باعث میشود حتی برای تعریف و توضیح دیتابیس، از مفاهیم و تعاریف RDBMS ها استفاده شود. شخصاً به یاد دارم که در کلاسهای پایگاه داده و آموزشهای ویدیویی، اولین مثالهای مرتبط با دیتابیس، دفترچه تلفن و جدولهای اکسل میباشد، که به معنای این است که نهتنها من، بلکه اکثر برنامهنویسها دید اول آنها از دیتابیس ، RDBMS ها و SQL است.

حال چه چیزی تعریف کردن NoSQL ها را سخت میکند؟ اولین و مهمترین علت نبود استاندارد لازم در حوزه دیتابیسهای NoSQL است. در بین رفرنسهای این مقاله، زمانی که در حال مطالعه کتاب Getting Started with NoSQL از Gaurav Vaish بودم، اصلاً به این مسئله فکر نمیکردم که شاید کتابهای دیگر بخواهند این دیتابیس را از دیدگاههای دیگر بررسی و تعریف کنند. یک علت دیگر نیز میتواند این باشد که در تعریف دیتابیسهای NoSQL گاهی از کمبودها و برتریهای آن نسبت به RDBMS ها استفاده میشود. شاید در نگاه اول چندان مسئله مهمی به نظر نرسد ولی اگر سرعت پیشرفت این دیتابیسها را در دهه گذشته در نظر بگیریم، این تعریف ناقص میشود. مثلاً در ابتدا زمانی که میخواستند NoSQL را تعریف کنند، دیتابیسهایی را NoSQL میگفتند که از SQL استفاده نمیکنند، اما مدتی بعد دیتابیسهای NoSQL پیدا شد که از زیرمجموعههای SQL و یا

زبان مشابه‌ای مانند SQL برای جست‌وجو اطلاعاتی استفاده می‌کردند. همچنین در بعضی تعریف‌ها از این دیتابیس، به آن‌ها دیتابیس‌هایی می‌گویند که از رابطه استفاده نمی‌کنند. شاید اکثر مواقع این تعریف چندان بد به نظر نرسد، اما اگر به پیاده‌سازی‌های دیتابیس‌های Object-Relation DB توجه کنیم، می‌بینیم که در این دیتابیس‌ها نیز سروکله روابط پیدا می‌شود، هرچند این رابطه‌ها شاید با RDBMS ها متفاوت باشد.

برای تعریف این دیتابیس کمی نیاز دارم تا اشاره‌ای به سیستم کاری RDBMS کنم و بعضی مفاهیم مرتبط نیز توضیح دهم.

سالیان سال دیتابیس‌های رابطه‌ای در حال ذخیره دیتاهایی هستند که آن‌ها را به‌عنوان دیتای ساختیافته می‌نامیم. این دیتاها به بخش‌های مختلفی تقسیم می‌شوند که به آن‌ها جدول Table می‌گوییم. جدول‌ها واحدهای خوش‌تعریف داده از نظر نوع، سایز و سایر محدودیت‌ها را در خود نگه می‌دارند. هر یک از واحدهای دیتا را ستون Column می‌گوییم و به هر واحد گروه ردیف Row می‌گوییم. ستون‌ها ممکن است دارای رابطه تعریف‌شده به روی ستون‌های دیگر داشته باشند تا بتوانیم این دیتاها را باهم مرتبط کرده و اطلاعات پیچیده‌تری از آن‌ها دریافت کنیم. به روشنی می‌توان گفت که در طراحی دیتابیس‌های RDBMS، شخص همواره با طراحی جدول‌ها و ستون‌ها سروکار دارد.

به توانایی اتصال دادن چندین نود سخت‌افزاری و نرم‌افزاری، به‌طوری‌که آن‌ها به‌صورت یک واحد منطقی کار کنند، Horizontal Scalability می‌گویند. مثلاً اضافه کردن یک واحد پردازشگر به خوشه‌های سرور برای بهبود عملکرد و یا اضافه کردن یک واحد

حافظه بلندمدت برای افزایش حجم سرور. اگر بخواهیم حجم دیتابیس خود را افزایش دهیم، یکی از روش‌ها افزایش آن‌ها به‌صورت Horizontal Scalability است که به علت تأکید بر روی سازگاری و صحت دیتا، Horizontal Scalability یک کار بسیار سخت است، اگر غیرممکن نباشد.

کمتر از ده سال گذشته، با پیدایش نرم‌افزارهای بزرگ تحت وب، تحقیقات زیادی انجام گرفت تا مدیریت دیتا در حجم بالا به‌خوبی انجام شود. یکی از خروجی‌های این تحقیقات، دیتابیس‌های غیر رابطه‌ای است که به‌طورکلی به آن‌ها NoSQL می‌گویند. یکی از مشکلاتی که این دیتابیس‌ها حل می‌کنند، Scale می باشد.

ویکی‌پدیا دراین‌باره می‌نویسد:

دیتابیس‌های NoSQL مکانیسم جدیدی برای نگه‌داری و خواندن دیتا دارند که اساس کار آن‌ها با RDB ها مختلف است. این دیتابیس‌ها از سال 1960 میلادی وجود داشتند اما تا اوایل قرن بیست و یکم موردتوجه قرار نگرفتند. این مدل دیتابیس‌ها ابتدا توسط شرکت‌هایی مانند Facebook، Google و Amazon استفاده شد. NoSQL دیتابیس‌ها به علت اینکه می‌توانند به‌راحتی از پس دیتاهای با حجم بسیار زیاد برآیند، بسیار در حوزه Big Data استفاده می‌شوند. سیستم‌های NoSQL گاهی Not Only SQL نیز خوانده می‌شوند تا بروی محدود نبودن به SQL و داشتن این زبان تأکید شود. استفاده آسان از آن‌ها، سادگی طراحی از دلایل استفاده ازاین‌گونه دیتابیس‌ها می‌باشد.

به نظر من ویکی‌پدیا تعریف بی‌طرفانه خوبی نسبت به این‌گونه دیتابیس‌ها دارد، اما بعد از مطالعه کتاب‌های مختلف، من دیتابیس‌های NoSQL را این‌گونه تعریف می‌کنم:

به دیتابیس‌هایی که برای یک هدف معین، ضرورتاً از تمامی قواعد و مفاهیم RDBMS ها استفاده نمی‌کنند، NoSQL می‌گویند.

برای مثال، هورویتز در مقاله‌ی مقدمه‌ای بر دیتابیس‌های شیگرا که در سال 1991 نوشته است می‌گوید که یکی از مشکلاتی که در استفاده از RDBMS ها وجود دارد، Impedance Mismatch هست. تمام برنامه نویسان زمانی که با دیتابیس‌های RDBMS ها کار می‌کنند، فاصله بین مقادیر داخل دیتابیس و داخل برنامه خود را حس می‌کنند. بخصوص زمانی که برنامه‌نویس‌ها از الگوهای شیگرایی در برنامه خود استفاده می‌کند. در برنامه‌نویسی شیگرا، کلاس‌ها سازه‌هایی هستند که می‌خواهند پیچیدگی‌ها را درون خود پنهان کنند اما گرفتن اطلاعات از دیتابیس و مقداردهی آن‌ها به اشیای برنامه، واقعاً کاری بسیار سخت و وقت‌گیر است و این پنهان‌کاری را از بین می‌برد. برای حل این مسئله، راه‌کارهای بسیار متفاوتی ایجادشده است مانند ساخت ORM، که در اصل یک واسط بین دیتابیس و کدهای شیگرای ما است اما مهم‌ترین آن‌ها ایجاد معماری‌های نرم افزاری مختلف و پیدایش دیتابیس‌های XML و شیگرا می‌باشد.

در اصل، با حذف و تغییرات بعضی مفاهیم دیتابیس‌های RDBMS مانند ساخت جدول و رکورد اطلاعاتی، Object Database ها برای حل این مشکلات پدیدار شدند.

جنبش دیتابیس‌های NoSQL در اوایل قرن بیست و یکم، زمانی که دنیا توجه خود را به روی دیتابیس‌های وب قرارداد، شروع شد. با اینکه ویکی‌پدیا به NoSQL، not only SQL می‌گوید، در ابتدا این کلمه به‌طور مستقیم تنها از دو کلمه No و SQL ساخته شد. چیزی که می‌خواهد این دو کلمه بگوید این بود که من نمی‌خواهم از SQL یا RDBMS استفاده کنم ولی برای راحتی به همان NoSQL اکتفا شده است.

بعضی نویسندگان دراین‌باره نوشته‌اند که NoSQL به انواعی از دیتابیس‌ها اشاره می‌کند که از اصول RDBMS بخصوص ACID استفاده نمی‌کند و طوری طراحی‌شده است که از پس اندازه اطلاعات Google،Facebook ، Yahoo، Twitter و خیلی‌های دیگر برآید. با اینکه این ابر شرکت‌ها بودند که باعث گسترده شدن این مدل دیتابیس‌ها شدند، سال 2014 شاهد پیدایش دیتابیس‌هایی بودیم که از مدل ACID نیز استفاده می‌کردند. برای مثال Neo4j یکی از گراف دیتابیس‌هایی است که از اصول ACID پیروی می‌کند.

همچنین در بعضی مقالات و کتاب‌ها نوشته‌شده است که دیتابیس‌های NoSQL برای اجراشدن بروی سرورها ساخته‌شده‌اند اما دیتابیس‌های گراف برای این کار ضرورتاً ساخته نشده‌اند. همچنین درحالی‌که می‌گویند NoSQL خارج از روابط است، دیتابیس‌های گراف برای دیتاهایی که روابط در آن‌ها بسیار مهم و پیچیده است طراحی‌شده است.

حال شاید شما هم مثل من از اینکه هر منبع و هر نویسنده یکطور سعی در توضیح و تعریف آن دارند، کلافه شده باشید. یکی از مهم‌ترین دلایل این شرایط این است که دیتابیس‌های NoSQL، یک نوع دیتابیس نیستند بلکه به مجموعه دیتابیس‌هایی

اشاره می‌کند که ضرورتاً از تمامی اصول RDBMS ها استفاده نمی‌کنند. مثلاً، همان‌طور که گفتم، Object Database ها و Graph Database ها از دو نوع مختلف دیتابیس NoSQL هستند که برای حل مسائل مختلفی طراحی شدند و به این معنی نیست که ضرورتاً از یک اصول پیروی می‌کنند.

برخی ویژگی‌های NoSQL

قبل از خواندن ویژگی‌های زیر، در نظر بگیرید که اکثر منابع ویژگی ها را به همه NoSQL ها ارتباط می دهند و این نادرست است. به همین خاطر من از لغت برخی استفاده کردم. البته اگر NoSQL را به عنوان مجموعه ای از دیتابیس ها مختلف بدانیم که حداقل یکی از آن ها از خاصیت های زیر برای آن ها صدق می کند، نیازی به لغت برخی نخواهد بود.

برخی از NoSQL ها چیزی بیشتر از ردیف‌های درون یک جدول می‌باشند. بعضی از آن‌ها اطلاعات را به‌صورت نودهایی درون یک گراف ذخیره می‌کنند. بعضی به‌صورت key-value می‌باشند و حتی بعضی از آن‌ها از خانواده ستون دیتا می‌باشند.

به‌طورکلی NoSQL به شما اجازه می‌دهد تا اطلاعات خود را بدون Join Query با استفاده از یک رابط بسیار راحت دریافت کنید.

اصولاً بدون شماتیک هستند. سیستم آن‌ها به شما اجازه می‌دهد که دیتای خود را درون یک فولدر کپی کنید و بدون کشیدن دیاگرام ER دیتا را از آن بیرون بکشید.

به‌راحتی به روی چند پروسه و پردازشگر قرار می‌گیرد. سیستم آن‌ها به شما کمک می‌کند که دیتابیس خود را روی چند پردازشگر قرار دهید و سرعت و عملکرد بالا را از دست ندهید.

وقتی پردازشگر بیشتری به سیستم خود اضافه می‌کنید، عملکرد دیتابیس شما افزایش پیدا می‌کند.

زبان NoSQL مشابه SQL نیست، ولی تعریف NoSQL نباید این محدودیت را در ذهن ایجاد کند که این‌گونه دیتابیس هیچ‌گونه

استفاده از زبان مشابه SQL ندارد یا از SQL استفاده نمی‌کند، بلکه در بعضی از این دیتابیس‌ها، SQL و زبان‌های دیگری برای جست‌وجوی اطلاعات استفاده می‌شود.

با اینکه خیلی از این دیتابیس‌ها Open Source هستند، اما نسخه‌های تجاری نیز دارد.

دیتابیس‌های NoSQL محدود به Big Data نیستند. اکثر این دیتابیس‌ها زمانی طراحی شدند که مشکل مدیریت کردن حجم‌های بزرگ دیتا زیاد به چشم می‌خورد. با اینکه حجم دیتا بسیار مهم است، NoSQL بر روی سرعت و تغییرات نیز تمرکز می‌کند.

درباره پردازش، ابری نیست. این‌گونه دیتابیس‌ها تنها برای پردازش ابری ساخته نشدند و می‌توان از آن‌ها درون دیتاسنترهای کوچک نیز استفاده کرد.

NoSQL محدود به سخت‌افزار خاصی نیست.

بهترین نوع دیتابیس برای سیستم‌های توزیع‌شده می‌باشند.

از هر متخصصی که بپرسید، به‌راحتی به شما می‌گوید که ACID مخفف چیست. این مفاهیم بیشتر از چهل سال است که وجود دارد و تا همین اواخر، مهم‌ترین معیار برای تمامی دیتابیس‌ها بود تا به آن دست بیابند. بدون ACID در یک سیستم، اعتماد و اطمینان از آن سیستم خارج می‌شد.

در دهه هفتاد میلادی، آقای Jim Gray به این ایده فکر کرد و درنهایت مقاله افسانه‌ای خود را به نام « The Transaction Concept: Virtues and Limitation» در سال 1981 منتشر کرد . شاید این اطلاعات چندان جدید نباشد و شما نیز هم‌اکنون کاملاً با داستان آن آشنا باشید. در طی نوشتن این قسمت، برایم جالب شد تا مقاله آقای Gray را مطالعه کنم.

چیزی که در مقاله ایشان نظرم را جلب کرد، این بود که آقای Gray تنها درباره مفاهیم Atomicity، Consistency و Durability صحبت کرده بود. به این معنا که در آن زمان، تأکیدی بر اینکه تراکنش‌ها نباید به تراکنش‌های دیگر دسترسی داشته باشند نبود.

این مقاله به سطوح تراکنش تمرکز کرده بود و تراکنش را یک قرارداد یا هر تعداد از «تغییر حالت سیستم» که یا باید اصول CAP را به‌عنوان خصوصیت سیستم به ارث ببرد یا محدودیت‌های سازگاری سیستم را تعیین کرد.

Bruce Lindsay و همکارانش در مقاله خود « Notes on Distributed Databases» در سال 1979 که از کار آقای Gray الهام گرفته بودند، اصول بنیادی برای دست یافتن به سازگاری «Consistency»

و استاندارد اولیه و مهم برای تکرار پایگاه «Database Replication» داده‌ها را ذکر کردند.

در سال 1983، Andreas Reuter و Theo Härder مقاله‌ای به نام «اصول بازیافت دیتابیس‌های تراکنش گرا» چاپ کردند و رسماً واژه ACID را ثبت کردند که تا دو دهه‌ی بعد به معنای زیر رسید:

Atomicity

یک کار یا همه کارها درون یک تراکنش، یا همگی انجام می‌شوند یا هیچ‌کدام انجام نمی‌شوند. این اصل همه یا هیچ است. اگر یکی از اصول تراکنش با موفقیت انجام نشود، تمامی تراکنش انجام نخواهد شد. بهترین مثالی که در این حوزه می‌توان بیان کرد، پرداخت‌های مالی می‌باشد؛ زیرا اگر پرداخت مالی را دو قسمت در نظر بگیریم، اگر یکی از قسمت‌ها درست انجام نشود، تعادل مالی به هم می‌خورد.

Consistency

هر تراکنش باید از تمامی پروتکل‌ها و قواعد توسط سیستم پیروی کند. تراکنش نباید این پروتکل‌ها را زیر پا بگذارد و دیتابیس در شروع و پایان تراکنش، باید دیتابیس را در حالت سازگار قرار دهد. هرگز چیزی به‌عنوان تراکنش نیمه انجام‌شده وجود ندارد.

Isolation

هیچ تراکنشی، به تراکنش دیگری که در حالت انجام و تمام نشده قرار دارد، نمی‌تواند دسترسی داشته باشد؛ بنابراین،

هر تراکنش تنها به خود وابسته است. این برای سازگاری و عملکرد تراکنش در دیتابیس ضروری است.

Durability

زمانی که تراکنش تمام شد، نتیجه تراکنش باقی می‌ماند و هیچگاه بازنمی‌گردد. از پس شرایط ازکارافتادن سیستم، قطع برق و مشکلات مشابه نیز برمی‌آید.

البته باید در نظر بگیرید که این تعاریف جنبه‌های مختلفی دارند و هر دیتابیس ممکن است هر یک از تعاریف را از جنبه‌ای متفاوت ببیند؛ اما در جهان RDBMS ها، ACID مهم‌ترین بخش اساس کار آن‌هاست و بدون آن، از صحت دیتا نمی‌توان اطمینان داشت. دیتابیس‌هایی که بر مبنای ACID کار می‌کنند، در هر میکروثانیه، تمام دیتاها را چک می‌کنند تا مطمئن شوند دیتایی محدودیت‌ها را نمی‌شکند. چنین سیستم‌هایی تاکنون برای سیستم‌های کوچک که نیاز به تغیر اندازه خاصی ندارند و نرمال‌سازی شده‌اند بسیار خوب کارکرده است؛ اما دیگر مانند گذشته چندان جوابگو نیست. دیتاهای ساخت نیافته، عظیم دیتا، ساختمان داده‌های غیر رابطه‌ای، پردازش توزیع‌شده و سازگاری نهایی در حال حاضر بسیار بیشتر مشاهده می‌شوند. نیازمندی‌های جدید، به‌معنی پیدایش فنّاوری‌های جدید است.

Base خودش را معرفی می‌کند.

خوشبختانه برای دنیای سیستم‌های پردازشگر توزیع یافته، مهندس‌های باهوشی وجود دارد. چگونه دنیای دیتای سیستم‌های بزرگ مانند BigTable گوگل، Dynamo آمازون و Cassandra فیس‌بوک با عدم وجود سازگاری، اعتماد و صحت سیستم را حفظ می‌کند؟ پاسخ، BASE است.

Basically Available

این محدودیت بیان می‌کند که سیستم موجود بودن و در دسترس بودن خود را تضمین نمی‌کند. برای هر درخواست، پاسخی خواهد بود اما پاسخ ممکن است عدم موفقیت باشد که بیان کند دیتا در حالت ناسازگار یا در حال تغییر باشد.

Soft State

حالت سیستم ممکن است در طول زمان تغییر کند. حتی زمانی که ورودی به سیستم وارد نمی‌شود، ممکن است به خاطر اینکه سیستم درنهایت باید در حالت سازگار باشد، در حال تغییر باشد. به همین علت حالت سیستم همواره نرم (Soft) است.

Eventual consistency

سیستم زمانی که ورودی نمی‌گیرد، درنهایت در حالت سازگار قرار می‌گیرد. دیتا در حالت‌های مختلف و وضعیت‌های دیگر دیر یا زود قرار می‌گیرد و گسترش می‌یابد، اما سیستم به دریافت

ورودی ادامه می‌دهد و سازگاری هر تراکنش را چک نمی‌کند مگر نه اینکه به تراکنش بعدی برود.

Base نسبت به ACID دارای انعطاف بیشتری نسبت به تغییر اندازه است و هزینه کمتری را داراست. چک کردن سازگاری تراکنش‌ها در هرلحظه فشار بسیار زیادی را به سیستم وارد می‌کند و هزینه‌ها را بسیار بالا می‌برد. مقدار پردازشی که برای این کار انجام می‌شود نجومی است. سازگاری نهایی به شرکت‌های بزرگی مانند یاهو و آمازون این امکان را می‌دهد که با هزینه کمتر، هم سازگاری را حفظ کنند و هم مشتری را راضی نگه‌دارند. البته با اینکه همواره سازگاری را حفظ خواهند کرد، اما این خود هزینه‌های دیگری داراست.

SQL vs NOSQL

	NoSQL	SQL
Model	Non-relational	Relational
	Stores data in JSON documents, key/value pairs, wide column stores, or graphs	Stores data in a table
Data	Offers flexibility as not every record needs to store the same properties	Great for solutions where every record has the same properties
	New properties can be added on the fly	Adding a new property may require altering schemas or backfilling data
	Relationships are often captured by denormalizing data and presenting it in a single record	Relationships are often captured in a using joins to resolve references across tables
	Good for semi-structured data	Good for structured data
Schema	Dynamic or flexible schemas	Strict schema
	Database is schema-agnostic and the schema is dictated by the application. This allows for agility and highly iterative development	Schema must be maintained and kept in sync between application and database
Transactions	ACID transaction support varies per solution	Supports ACID transactions
Consistency	Consistency varies per solution, some solutions have tunable consistency	Strong consistency supported
Scale	Scales well horizontally	Scales well vertically

قبل از اینکه بخواهیم RDBMS ها را با NoSQL ها مقایسه کنیم، باید در نظر بگیرید که این دیتابیس‌های جدید برای حل مسئله ای ساخته‌شده اند که RDBMS ها به‌راحتی نمی‌توانند آن‌ها را حل کنند. بنابراین درست نیست که بخواهیم همه آن‌ها را با RDBMS ها مقایسه کنیم، اما می‌توانیم نکاتی را ذکر کنیم.

ابتدا اجازه دهید عکس بالا را که از سایت ماکروسافت گرفته ام توضیح دهم.

مدل

ازنظر مدل، SQL دیتابیس از مدل رابطه‌ای استفاده می‌کنند و دیتا را در جدول‌ها نگه می‌دارند، اما NoSQL ها مدل کاملاً متفاوتی دارند و اکثر آن‌ها، از مدل غیر رابطه‌ای استفاده می‌کنند. حتی اگر رابطه نیز استفاده کنند، این رابطه کاملاً با رابطه‌های RDBMS ها متفاوت است. شیوه نگهداری دیتا نیز با SQL متفاوت است.

دیتا

SQL ها برای دیتاهایی ساخته‌شده که هر رکورد دارای خصوصیت‌های ثابتی می‌باشد. اگر لازم باشد که تغییراتی در خصوصیت‌های رکوردها ایجاد شود، به معنای حذف یا اضافه ستون‌ها، مجبور به تغییرات شماتیک دیتابیس خواهیم بود و کلاً این کار چندان راحت نیست زیرا ساختار نرم‌افزارها را نیز ممکن است تغییر دهد. با استفاده از Join، می‌توان از رابطه بین جدول‌های مختلف استفاده کرد.

NoSQL ها این امکان را عموماً به شما می‌دهند که مجبور نباشید که برای اضافه کردن خصوصیت‌ها شماتیک را تغییر دهید. بدون اینکه زحمتی بکشید، می‌توانید به یک دیتا خصوصیتی اضافه یا کم کنید. درحالی‌که نرمال‌سازی یک اصلی مفید در دیتابیس‌های رابطه‌ای است، از عکس نرمال‌سازی در دیتابیس‌ها NoSQL استفاده می‌شود تا از روابط بین رکوردها استفاده شود.

اگر دیتاهای ما ساختیافته است، دیتابیس‌های SQL بهتر از NoSQL ها کار می‌کنند.

شماتیک

دیتابیس‌های NoSQL اصولاً بدون شماتیک خاصی هستند به این معنا که معمولاً کاربر دیتابیس شماتیک دیتابیس را مشخص می‌کند و یا تغییر می‌دهد؛ اما شماتیک دیتابیس درون دیتابیس‌های رابطه‌ای همواره ثابت و به‌راحتی قابل‌تغییر نمی‌باشد. یعنی اگر شماتیک درون دیتابیس‌های رابطه‌ای تغییر کند، نرم‌افزارها نیز باید از این تغییر باخبر شوند.

تراکنش

در حوزه تراکنش، به نظر می‌رسد بهترین دیتابیس‌ها برای کنترل و مدیریت دیتابیس که بر اساس ACID کار کنند، دیتابیس‌های رابطه‌ای می‌باشند؛ اما درون دیتابیس‌های NoSQL دیتابیس‌های وجود دارد که بر اساس ACID کار کنند.

سازگاری

یکی از اصول اصلی دیتابیس‌های رابطه‌ای، سازگاری است و این دیتابیس‌ها سازگاری را تضمین می‌کنند؛ اما دیتابیس‌های NoSQL نسبت به سازگاری متفاوت می‌باشند. بعضی از آن‌ها چون بر اصول ACID نوشته‌شده‌اند آن را پشتیبانی می‌کنند و بعضی دیگری که بر اساس BASE ساخته‌شده‌اند، سازگاری نهایی را دارند.

تغییر اندازه

دیتابیس‌های SQL چندان توزیع‌پذیر نیستند و نمی‌توان به‌خوبی از توزیع‌پذیری به همراه این دیتابیس‌ها استفاده کرد، اما دیتابیس‌های NoSQL عموماً دارای توزیع‌پذیری خوبی هستند.

حال اجازه دهید تا درباره انتخاب بین این دو نیز اطلاعاتی به شما بدهم. اگر شما کلاً به مفهوم دیتابیس ناآشنا هستید یا تازه آن را شروع کرده‌اید، دیتابیس‌های رابطه‌ای را انتخاب کنید. اولین علت این هست که این دیتابیس‌ها بسیار پخته هستند و جامعه آموزشی قوی دارند و اکثریت برنامه‌نویس‌ها به آن‌ها مسلط هستند. شما به‌راحتی می‌توانید یک نفر را پیدا کنید تا به شما در مشکلات کمک کند. ابزارهای بسیار زیادی بر روی دیتابیس‌های رابطه‌ای ساخته‌شده که شما می‌توانید از آن‌ها استفاده کنید. انتخاب کردن یک فنّاوری جدید شاید یک ریسک نیز باشد. همچنین بیشتر تحلیل‌های نرم‌افزاری بر اساس دیاگرام ER می‌باشد و می‌توانید از آن‌ها مستقیماً استفاده کنید.

زمانی از دیتابیس‌های NoSQL استفاده کنید که می‌دانید برای چه چیزی امکانات و ابزارهای دیتابیس‌های رابطه‌ای را فدای آن می‌کنید. مثلاً اگر عملکرد و سرعت دیتابیس‌های رابطه‌ای برای شما کافی نباشد یا حجم دیتابیس شما به حدی می‌رسد که مجبور باشید دیتابیس خود را توزیع کنید. توجه کنید که هریک از دیتابیس‌های NoSQL دارای قدرت و محدودیت‌های مختص خود است.

اگر دوست ندارید چندان درگیر کار با ذخیره‌سازی اطلاعات برنامه خود باشید و ساخت و مدیریت دیتا درون دیتابیس‌های رابطه‌ای شما را اذیت می‌کند، می‌توانید از دیتابیس‌های شی گرا استفاده کنید.

همچنین یکی از مسائلی که شاید در استفاده از دیتابیس NoSQL به نظر برسد این هست که زمانی که برای ساخت مدل پروژه انجام می‌شود، بسیار کمتر از دیتابیس‌های SQL هست. با این‌حال در نظر بگیرید که نمی‌توان برای تمامی دیتابیس‌های NoSQL این حرف را تصدیق کرد. زیرا اول اینکه من به‌تمامی دیتابیس‌های NoSQL تسلط ندارم و دوم شاید دیتابیس‌هایی وجود داشته باشد یا در آینده به وجود بیاید که پیاده‌سازی مدل با آن‌ها در عمل زمان‌بر باشد.

عملکردهای دیتابیس‌های NoSQL و RDBMS ها در نرم افزار تراکنش گرا

این نرم افزارها دیتاهایی دارند که ماهیتاً، دارای رابطه با یکدیگر هستند. خصوصیت‌های مهم این نرم افزارها بدین‌صورت است:

نرم افزار اهمیت ویژه ای به سازگاری و یکپارچگی دیتا می‌دهد.

کاربرد دسترسی موازی به نسبت پایین‌تر است. البته این به این معناست که نرم افزار با یک نسخه از دیتابیس سروکار دارد و نیاز به تکرار دیتابیس «database replication» و متعادل‌کننده «Load-balance» ندارد.

نرم افزارهای تجاری یک مدل ازاین‌گونه نرم افزارها می‌باشد.

برای نرم افزارهای تراکنش گرا، نیاز عمومی شماتیک دیتابیس به‌صورت زیر است:

- تعریف دقیق و قوی ساختیافته از خصوصیت‌ها، انواع و قانون‌های دیتابیس.
- توانایی تعریف دقیق رابطه‌ها.
- شماتیک در طول زمان چندان تغییر نمی‌کند.

از دید دسترسی اطلاعاتی، نیازمندی‌های زیر وجود دارد:

- سازگاری: هر خواندن دیتا، باید جدیدترین دیتا را بازگرداند.
- اکثر مواقع، کل رکورد خوانده می‌شود.
- خواندن و دسترسی به دیتا در چند جدول متداول است.

برای این‌گونه نرم افزار دو گونه دیتابیس به نظرمی تواند به این مسئله کمک کند:

- دیتابیس‌های ستون گرا به شما کمک می‌کند تا ساختار را تعریف کنید. اگر لازم باشد، می‌توان آن را در طول زمان تغییر داد.
- دیتابیس‌های سند گرا به شما کمک می‌کند تا «View» بسازید یا «Join» را پیاده‌سازی کنید.

برای نرم‌افزارهای تراکنش گرا، NoSQL دارای محدودیت‌های زیر است:

- عدم توانایی تعریف روابط
- نبود تراکنش‌ها
- عدم وجود خصوصیت ACID
- اکثر NoSQL ها قدرت خود را در یک سیستم نشان نمی‌دهند.
- عدم وجود دستورات JOIN

نتیجه

برای این‌گونه نرم‌افزارها، RDBMS ها انتخاب بهتری هستند.

تاریخچه NoSQL

A history of databases in No-tation:

1970:NoSQL = We have no SQL

1980:NoSQL = Know SQL

2000:NoSQL = No SQL!

2005:NoSQL = Not only SQL

2013:NoSQL = No,SQL!

Minson

همانند اکثر فنّاوری‌های جدید، NoSQL نیز همراه با ترس، عدم قطعیت و شک است. در مقابله با NoSQL دنیای برنامه نویسان به سه دسته کلی تقسیم می‌شود:

- کسانی که آن را دوست دارند.

افراد در این گروه در حال جستوجو درون NoSQL هستند تا بفهمند این دیتابیس چگونه در برنامه‌ها قرار می‌گیرد. آن‌ها از آن استفاده می‌کنند، آن را می‌سازند و با آن همگام می‌شوند.

- کسانی که آن را رد می‌کنند.

افراد این گروه معمولاً به روی کمبودهای NoSQL تأکید می‌کنند و سعی می‌کنند بگویند آن‌ها بی‌ارزش هستند.

- کسانی که به آن اهمیت نمی‌دهند.

در این گروه برنامه نویسان یا منتظر هستند این فنّاوری پخته شود یا هنوز وقت کافی نداشته‌اند که به آن رسیدگی کنند.

اگر شما از دسته افرادی هستید که به آن اهمیت نمی‌دهید یا عاشق آن هستید، شاید برای شما جالب باشد که بدانید NoSQL از کجا شروع شد.

NoSQL فنّاوری واحدی نیست که توسط چند نفر داخل یک گاراژ اختراع‌شده باشد یا نظریه‌های ریاضیاتی خاصی درباره داده ساختارها باشد. مفاهیم پشت NoSQL بسیار آرام در طی چندین سال توسعه یافت. افرادی غیر وابسته به سازمان و یا اداره خاصی آن‌ها را مورد بررسی قرار دادند و برای حل مشکلات خود از آن‌ها استفاده کردند که باعث شد دیتابیس‌های NoSQL مختلفی ساخته شود.

اولین باری که رسماً از لغت NoSQL استفاده شد، در سال 1998 توسط Carlo Strozzi بود. وی در حال دیدن شهر San Francisco بود و می خواست که دیتابیس سبک و رابطه ای خود را به چند نفر نشان دهد. سیستم مدیریت پایگاه داده رابطه ای (RDBMS) دیتابیس‌های جا افتاده و پایدار کنونی هستند. اگر شما از یک دانشمند کامپیوتری که در 20 سال گذشته تحصیلات خود را به پایان رسانده است بپرسید که دیتابیس چیست، احتمال زیاد او یک دیتابیس رابطه‌ای را به شما توضیح خواهد داد.

کارلو از واژه NoSQL استفاده کرد زیرا دیتابیس او توسط اسکریپت‌های Shell کنترل می‌شد و از زبان ساختیافته پرس‌وجو (SQL) استفاده نکرد. معنای اصلی NoSQL «نه SQL» می‌باشد. به این معنا که به‌جای استفاده از SQL، مکانیسم پرس‌وجویی

استفاده کرد که بیشتر مشابه محیط پیاده‌سازی برنامه نویسان بود که برای کارلو، دنیای اسکریپت UNIX بود.

استفاده از این واژه، اولین چیزی که نشان می‌دهد ناامیدی برنامه نویسان از SQL است. با اینکه جمعیت بسیاری در حال حاضر مسلط بر SQL هستند و آموزش‌های خوبی برای دیتابیس‌های رابطه‌ای در حال حاضر وجود دارد، واژه NoSQL گرایش پیدا کردن راه بهتر را نشان می‌دهد، یا حداقل راه بهتری است برای برنامه نویسان تنبل که حوصله یادگیری و سر کله زدن با جمله‌های SQL طولانی و پیچیده را ندارند.

ملاقات کارلو در شهر San Francisco آمد و گذشت. توسعه‌دهندگان ازآن‌پس به ساخت و تست کردن مکانیسم‌های متفاوتی پرداختند. فنّاوری‌های متفاوتی ایجاد شد تا پیچیدگی‌های SQL را از دید توسعه‌دهندگان پنهان کنند. Hibernateبرای جاوا و Entity Framework از مهم‌ترین فنّاوری‌هایی بودن که به‌عنوان یک واسط بین دیتابیس و کاربر قرار می‌گرفتن و کار را تا حدی برای برنامه نویسان آسان می‌کردند.

استفاده از SQL هزینه‌بر است. Query های پیچیده به‌راحتی دیباگ نمی‌شوند و به‌راحتی می‌توان در آن‌ها تغییرات ایجاد کرد تا بهتر عمل کنند، به همین علت زمان انجام پروژه، مدیریت و آزمودن آن بسیار سخت است. پیدا کردن یک لایبری یا یک مکانیسم متفاوت برای پنهان کردن این پیچیدگی‌ها، حداقل یک‌راه خوبی برای کاهش هزینه‌ها است و استفاده از بهترین روش‌ها را میسر می‌سازد.

با این روش‌ها، به تنها چیزی که می‌رسید، پنهان کردن پیچیدگی‌هاست. درنهایت مشکلات دیتایی ایجاد می‌شود که

نیازمند یک روش کاملاً متفاوت تفکر نیاز دارد. فنّاوری رابطه‌ای کنونی چندان با چنین مشکلاتی کنار نیامد و انفجار وب این مشکلات را چند برابر کرد.

در سال 2006، گوگل مقاله‌ای را منتشر کرد که دیتابیس توزیع‌پذیر عظیم جدول (Bigtable) ساختیافته خود را توضیح می‌داد. گوگل عظیم جدول‌ها را این‌گونه توصیف کرد:

عظیم جدول سیستم ذخیره‌سازی توزیع‌پذیری برای مدیریت کردن دیتای ساختیافته به‌طوری‌که برای تغییر سایز دادن به اندازه بسیار حجیم مانند پتابایت به روی هزاران سرورهای عادی طراحی‌شده است.

در نگاه اول، عظیم جدول‌ها هم مانند RDBMS ها ردیف‌ها را با یک کلید ردیف و دیتا را در ردیف‌ها بین خانواده‌ای از ستون‌های اطلاعاتی ذخیره می‌کند. عظیم جدول‌ها مجموعه‌ای از کلید و مقدار (Value،Key) هایی هستند که هر کلید یک ردیف را مشخص و می‌کند و هر مقدار مجموعه‌ای از ستون‌ها هم‌خانواده را (Column Families).

در سال 2007، آمازون مقاله خود را درباره نرم‌افزار ذخیره اطلاعات داینامو (Dynamo) منتشر کرد. به زبان آمازون:

داینامو برای مدیریت حالت (State) های سرویس‌هایی که نیازمند اطمینان بسیار زیاد هستند و مدیریت تعادل بین در دسترس بودن، ثبات و صحت، به‌صرفه بودن و کارایی می‌خواهند، ساخته‌شده است.

در ادامه این مقاله توضیح می‌دهد که چگونه دیتاهای آمازون توسط کلید اصلی ذخیره‌شده و چگونه از هش (Hash) کردن سازگار

برای توزیع‌پذیری دیتا استفاده‌شده و نحوه استفاده از ورژن بندی بر اساس شی (Object Versioning) برای نگهداری صحت و سازگاری دیتا به روی دیتاسنترها توضیح داده‌شده.

اساس کار آمازون بسیار ساده بود، درواقع آن‌ها اولین دیتابیسی را پایه‌گذاری کردن که بر اساس یک کلید منحصر به‌فرد و یک مقدار که هر چیزی ممکن بود باشد، کار می‌کرد.

تا سال 2009، دیتابیس‌های NoSQL بسیار زیاد متن بازی (Open Source) پدیدارشاند. Riak ، MongoDB، HBase، Accumulo، Hypertable، Redis، Cassandra، Neo4j از مجموع دیتابیس‌هایی بودند که بین سال‌های 2007 و 2009 ساخته شدند.

این تغییرات محیطی بسیار سریع باعث شد که Eric Evans از Rackspace و Johan Oskarsson از Last.fm تا اولین جلسه مدرن NoSQL را برگزار کنند. برای اینکه محتوای این جلسه به‌راحتی بین جامعه‌های مجازی پخش شود، آن‌ها از هشتک #NoSQL استفاده کردند. این جلسه از اولین قدم برای استانداردسازی دیتابیس‌های NoSQL بود.

انواع دیتابیس‌های NoSQL

در این بخش قصد دارم درباره انواع دیتابیس‌های NoSQL بنویسم و توضیح دهم هرکدام چه مزیت‌هایی دارند و استفاده از آن‌ها چه چیزهایی را در RDBMS ها قربانی می‌کند.

دیتابیس‌های ستون گرا (Column-oriened databases)

دیتابیس‌های ستون گرا، برخلاف RDBMS ها، اطلاعات را بجای ذخیره کردن در ردیف‌ها، در ستون‌ها ذخیره می‌کنند. این مدل دیتابیس‌ها از زمان پیدایش خود مفهوم دیتابیس‌ها وجود داشته‌اند. TAXIR، نرم‌افزاری که در حوزه زیست‌شناسی بود، اولین نرم‌افزاری بود که از دیتابیس‌های ستون گرا استفاده می‌کرد.

دیتابیس‌های رابطه‌ای، دیتا را به‌صورت جدول‌های دوبعدی تشکیل‌شده از ستون‌ها و ردیف‌ها نشان می‌دهند که دریافت و نوشتن دیتا در هر زمان‌بر روی یک ردیف انجام می‌شود، درحالی‌که دیتابیس‌ها ستون گرا دیتا را در ستون‌ها ذخیره می‌کنند. برای مثال فرض کنید دیتاهای زیر قرار است ذخیره شود:

کد کارمندی	نام	نام خانوادگی	سن	حقوق
191	حسین	احمدی	45	1500000
229		رضاییان	34	1000000
881	احسان	محمدی	39	7000000
421	نوید	شناسا	32	2000000

در RDBMS ها اطلاعات به شکل زیر سریال شده «Serialized» و ذخیره‌سازی می‌شوند:

191، حسین، احمدی ، 45، 1500000

229، رضاییان ، 34، 1000000

881، احسان، محمدی ، 39، 7000000

421، نوید، شناسا ، 32، 2000000

اما در دیتابیس‌های ستون گرا، دیتا به فرم زیر ذخیره می‌شوند:

421، 881، 229، 191

حسین، احسان، نوید

احمدی، رضاییان، محمدی، شناسا

45، 34، 39، 32

1500000 ،1000000 ،7000000 ،2000000

البته در نظر بگیرید که این‌گونه نمایش بسیار ساده‌سازی شده است. دیتابیس‌ها معمولاً سیستم‌های بهینه‌سازی خاصی برای ذخیره‌سازی اطلاعات دارند. این نمایش تنها برای بیان و توضیح دیتابیس‌های ستون گرا استفاده شده است.

اکثر دیتابیس‌های ستون‌گرا، اجازه می‌دهند که در طول زمان بدون نگرانی از مقدارهای اولیه ردیف‌ها، ستون‌های جدیدی را بسازید. این موضوع انعطاف‌پذیری خاصی در طراحی و مدل می‌دهد، بدین‌صورت که اگر ستون خاصی را اضافه نکردید، در آینده به‌راحتی بتوانید آن را اضافه کنید.

برتری خاصی در استفاده از زیر مجموعه‌ای از ستون‌های موجود وجود دارد. برای مثال، بر روی مجموعه دیتای بزرگ، در محاسبه بزرگترین، کمترین، میانگین و مجموع دیتابیس ها ستون گرا، عملکرد بسیار درخشانی دارند.

به طور مشابه، زمانی که مقادیر جدیدی برای یک یا تمام ردیف ها اعمال می شود، این دیتابیس ها اجازه دسترسی چند تکه به دیتا می دهد به طوری که نیازی به دسترسی به ستون های بی ربط در محاسبه نباشد.

دیتابیس های سند گرا (Document Oriented Store)

دیتابیس‌های سند گرا، اجازه دسترسی و مدیریت دیتاهای نیمه ساختیافته را می‌دهد. بیشتر دیتابیس‌های این مجموعه، از BSON،JSON،XML و یا YAML برای ذخیره‌سازی دیتا استفاده می‌کنند. در مقایسه با RDBMS ها، هر سند «Document» نقش یک رکورد یا ردیف اطلاعاتی را دارد اما در مقایسه با RDBMS ها، این دیتا نیمه ساختیافته است.

برای مثال، دو رکورد ممکن است کاملاً فیلد و ستون‌های کاملاً متفاوتی داشته باشد. رکوردها ممکن است شماتیک خاصی نداشته باشند، درحالی‌که در RDBMS ها، تعریف جدول‌ها بخش ضروری از ساخت دیتابیس است. به همین خاطر، دیتابیس‌های سند گرا ممکن است شماتیک را پشتیبانی نکند و اصلاً هیچ محدودیت شماتیک را برای دیتا قرار ندهد.

با اینکه این دیتابیس‌ها شماتیک خاصی را ساپورت نمی‌کنند، اما اندیس‌ها (Index) ها را می‌توان ساخت و جستوجو کرد.

در این دسته، به دیتابیس‌های زیر می‌توان اشاره کرد:

MongoDB

CouchDB

Jackrabbit

Lotus Notes

Terrastore

BaseX

مهم‌ترین مزیتی که این دیتابیس‌ها نسبت به RDBMS ها دارند، نبود یا ضروری نبودن شماتیک دیتاها می‌باشد. این مسئله در نرم‌افزارهای وب زمانی که نیاز به ذخیره‌سازی دیتاهایی از انواع مختلف که در طول زمان ممکن است تغییر کنند، بسیار کاربردی است. برای مثال، در فروشگاه‌های اینترنتی، اطلاعات درباره کاربرها، انبار و سفارش‌ها را می‌توان به‌صورت سندهای ساده‌ی JSON یا XML ذخیره کرد. توجه کنید که ذخیره‌سازی سند گرا با «blob store» کاملاً متفاوت است زیرا این نوع ذخیره‌سازی، نمی‌توان اندیس گذاری کرد.

جستوجو کردن در بین نوع‌های دیتا، در مقایسه با RDBMS ها و دیتابیس‌های ستون‌گرا، بسیار سریع‌تر است. علت این هست که دیگر مفهوم جدول وجود ندارد، در اصل دیتابیس بدون شماتیک را بیان می‌کند. هر کس می‌تواند بدون در نظر گرفتن شماتیک، رکوردهایی را جستوجو کند.

دیتابیس‌های کلید- مقدار (Key-value store)

این دسته دیتابیس‌ها بسیار مشابه دیتابیس‌های سند گرا هستند. به این صورت که اجازه ذخیره‌سازی دیتا را بر اساس یک کلید می‌دهند. مشابه دیتابیس‌های سند گرا، نیازی به تعریف شماتیک برای یک مقدار نیست. اما، چند محدودیت توسط این دیتابیس‌ها به روی دیتا اعمال می‌شود:

- برخلاف دیتابیس‌های سند گرا که در زمان اضافه کرد یک سند به دیتابیس یک کلید ساخته می‌شود، ذخیره‌سازی کلید-مقدار نیازمند این هست که این کلید منحصربه‌فرد باشد.
- برخلاف دیتابیس‌های سند گرا که مقدار می‌تواند اندیس گذاری و جست‌وجو شود، در دیتابیس‌های کلید- مقدار، از کلید برای جست‌وجو اطلاعات اضافه می‌شود.

برخی از دیتابیس‌های این قسمت به شرح زیر است:

Redis

Memchached

MemchaeDB

Berkley DB

Voldemort

دیتابیس‌های کلید- مقدار، برای جست‌وجو بر اساس کلید بهینه‌شده‌اند. برای همین، آن‌ها به‌خوبی از حافظه کش (cache) استفاده می‌کنند. برای مثال، Redis به شما اجازه می‌دهد که فهرستی از تمامی کلیدهایی که از یک الگوی خاص پیروی می‌کنند را به شما بدهد. با اینکه پیچیدگی زمانی برای یافتن کلید بر اساس یک الگو، خطی می‌باشد (O(n))، مقدار ثابت آن بسیار کم و زمان آن بسیار پایین است. مثلاً برای یک

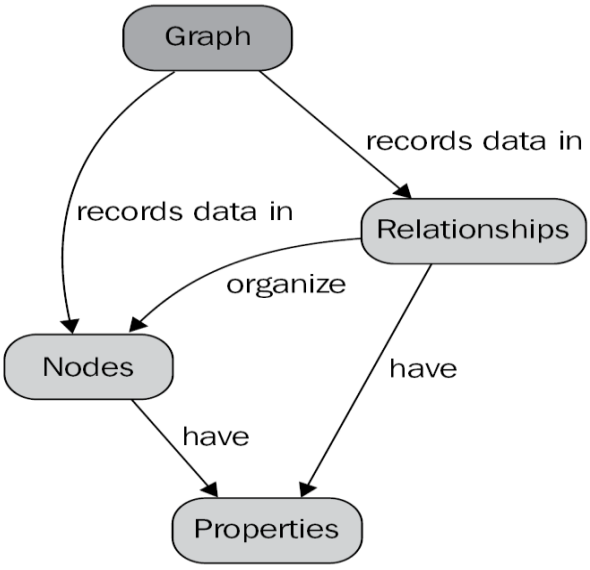
لب تاپ معمولی و ساده، Redis می تواند یکمیلیون کلید را در چهل میلی‌ثانیه اسکن کند.

با اینکه دیتابیس‌های کلید- مقدار نمی‌توانند بر اساس مقدار جستوجو کنند، آن‌ها می‌توانند نوع داده آن‌ها را بفهمند. به همین خاطر، بر اساس نوع داده‌ها، امکانات بسیار قویی ایجاد می‌شود.

دیتابیس‌های گراف (Graph Database)

در طراحی دیتابیس‌های RDBMS، گاهی وقت‌ها که روابط بین جداول بسیار مهم و پیچیده می‌شود، کار طراحی آن کمی دشوار می‌شود. گراف دیتابیس‌ها بخشی از دیتابیس‌های NoSQL هستند که روابط به‌صورت گراف‌ها نمایش داده می‌شود. بین هر نود در گراف، ممکن است چندین رابطه وجود داشته باشد که بیانگر روابطی است که آن‌ها به اشتراک می‌گذارند.

مثلاً این روابط می‌تواند رابطه اجتماعی بین افراد جامعه، یا ارتباطات خیابان‌ها و یا شبکه باشد.



ازجمله دیتابیس‌های معروف این قسمت می‌توان دیتابیس‌ها زیر را نام برد:

Neo4j

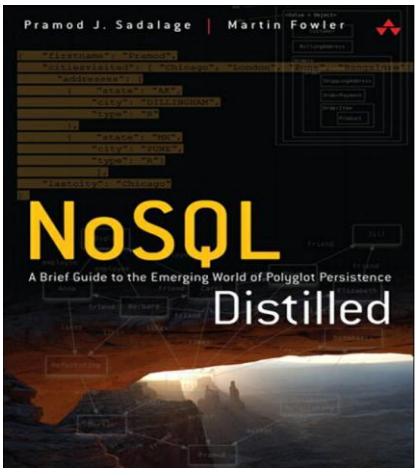
FlockDB

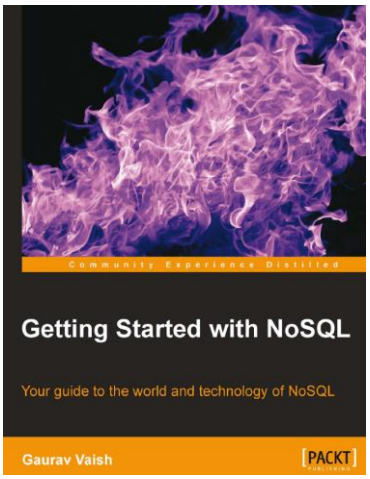
دیتابیس‌های شی گرا (Object Databases)

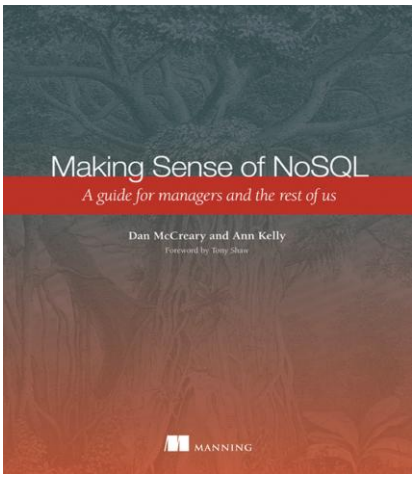
دیتابیس‌های شی گرا، دیتابیس‌هایی هستند که اطلاعات را به‌صورت شی (Object) ذخیره‌سازی می‌کنند. دیتابیس‌های شی گرا، قدرت دیتابیس را با اصول شی گرایی یکپارچه می‌کنند. این‌گونه دیتابیس‌ها از دهه هفتاد و هشتاد میلادی وجود داشته‌اند و بحث‌های زیادی درباره آن‌ها شده است. این دیتابیس ها، بخاطر راحتی در استفاده از آن‌ها، اولین دیتابیس مورد علاقه من است. نکات زیادی درباره این سبک دیتابیس هست که خود نیاز به یک کتاب مستقل برای توضیح آن هاست. اما اگر مایل به مطالعه درباره آن ها هستید، می توانید با مقاله هورویتز درباره دیتابیس های شی‌گرا شروع کنید.

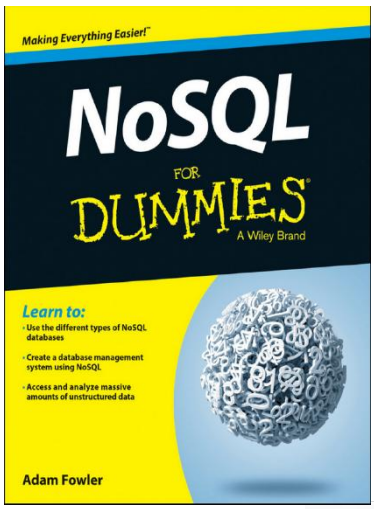
دیتابیس های NoSQL هرچند انقلابی هستند، باید توجه کرد که بسیار جوان هستند و برای حل تمامی مسئله‌ها ساخته نشده‌اند. قبل‌از اینکه یکی از دیتابیس‌های NoSQL را انتخاب کنید، بررسی کنید که چه نیازی در برنامه‌ی شما وجود دارد که دیتابیس‌های رابطه‌ای سنتی آن‌را برآورده نمی کند.

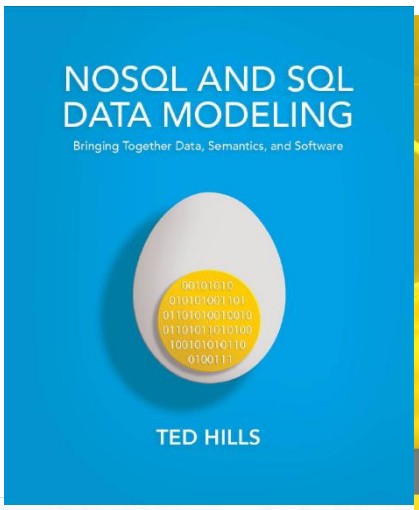
منابع

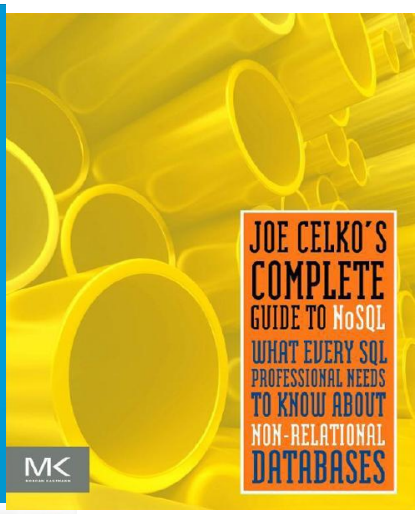


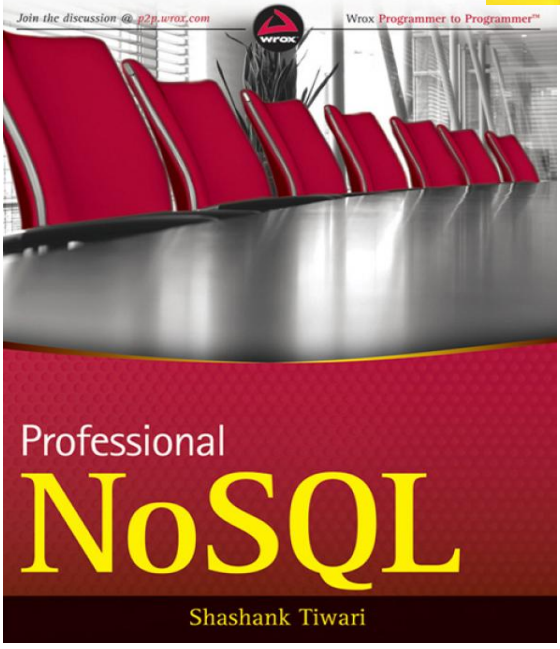












https://neo4j.com/blog/acid-vs-base-consistency-models-explained/

http://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/

https://en.wikipedia.org/wiki/NoSQL

http://nosql-database.org/

https://www.mongodb.com/nosql-explained

http://www.couchbase.com/nosql-resources/why-nosql

https://en.wikipedia.org/wiki/Object_database

https://en.wikipedia.org/wiki/Object-relational_impedance_mismatch