NoSQL is het nieuwe zwart. Ernstig in de mode. Misschien wordt NoSQL wel het nieuwe SOA. Ik hoop het niet, want dan moet iedereen het hebben. Dat zou enorm veel nodeloze complexiteit opleveren. Toch is er een merkbare opkomst van NoSQL en gerelateerde technologie. De vraag is: wanneer gebruik je dat? Wanneer gaan we voor NoSQL?

NoSQL is echt een volwassen technologie

Maar wanneer gaan we voor dit alternatief?

Allereerst is het goed om vast te stellen dat NoSQL helemaal niets nieuws is. Het inzetten van iets anders dan een relationele database voor opslag gebeurt al sinds jaar en dag. Wie heeft er bijvoorbeeld wel eens full text search geïmplementeerd op de volgende manier?

```
SELECT * FROM CONTENT C WHERE C.CONTENTBODY
LIKE '%zoekterm%'
```

Als het goed is niet, want daar is een database niet voor gemaakt. *Like queries* zijn niet efficiënt en schalen lastig. Daarnaast bieden ze niet de functionaliteit die nodig is voor echte full text search. Beter is het om bijvoorbeeld Apache Lucene in te zetten, want dat is werkelijk een persistente full text zoekindex met alle functionaliteit en performance die je daarvan mag verwachten. NoSQL dus.

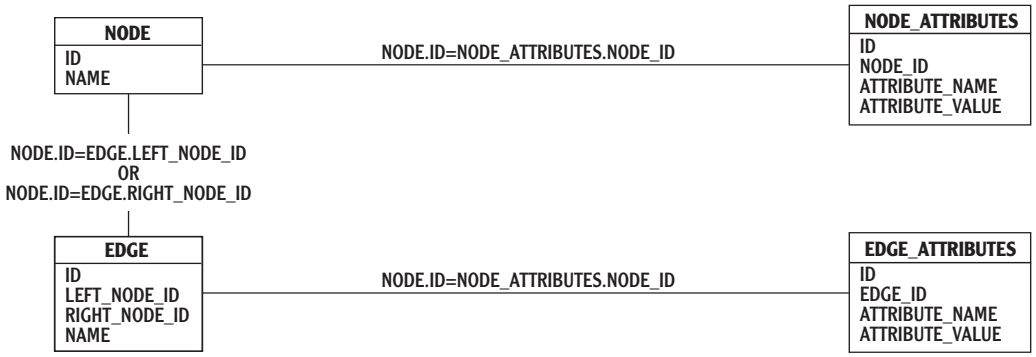
Onze favoriete hamer

Heb je wel eens geserializeerde Java-objecten weggeschreven naar disk? NoSQL. Gebruik je XML en XPath expressies om bepaalde informatie op te slaan en terug te halen? NoSQL. Kortom, het idee om iets anders dan een relationele database te gebruiken is niet nieuw. Desondanks laten we ons wel snel verleiden tot het gebruik van een SQL database op het moment dat we met iets meer data te maken krijgen, terwijl dat misschien niet altijd de beste oplossing is.

Stel je het volgende model voor om een graaf op te slaan in een database (zie figuur 1).
Een graaf heeft nodes (vertices) en edges. Zowel een node als een edge kunnen arbitraire attributen hebben. Een edge verbindt altijd twee nodes (LEFT en RIGHT).

Om nu gegeven een bekende node met ID = 42 alle omringende nodes op te vragen, schrijf je misschien deze query:

```
SELECT * FROM NODE N, EDGE E WHERE
(N.ID = E.LEFT_NODE_ID AND E.RIGHT_NODE_ID = 42)
OR (N.ID = E.RIGHT_NODE_ID
AND E.LEFT_NODE_ID = 42)
```



Figuur 1: Model om een graaf op te slaan in een database.

Friso van Vollenhoven
is NoSQL specialist bij Xebia.

Dat is te doen, maar wat gebeurt er als je één niveau verder wil kijken?

```
SELECT * FROM NODE N, EDGE E WHERE (E.LEFT_NODE_ID
IN (SELECT FIRSTNODE.ID FROM NODE FIRSTNODE,
EDGE FIRSTEDGE WHERE
(FIRSTNODE.ID = FIRSTEDGE.LEFT_NODE_ID
AND FIRSTEDGE.RIGHT_NODE_ID
= 42) OR (FIRSTNODE.ID = FIRSTEDGE.RIGHT_NODE_ID
AND FIRSTEDGE.LEFT_NODE_ID = 42))
AND E.RIGHT_NODE_ID = N.ID)
OR (E.RIGHT_NODE_ID IN (SELECT FIRSTNODE.ID
FROM NODE FIRSTNODE, EDGE FIRSTEDGE
WHERE (FIRSTNODE.ID = FIRSTEDGE.LEFT_NODE_ID
AND FIRSTEDGE.RIGHT_NODE_ID = 42)
OR (FIRSTNODE.ID = FIRSTEDGE.RIGHT_NODE_ID
AND FIRSTEDGE.LEFT_NODE_ID = 42))
AND E.LEFT_NODE_ID = N.ID)
```

Niet fraai. Zeker ook niet efficiënt. Veel zoekvragen zijn ook niet met een enkele query op te lossen. Dit is niet waar een relationele database sterk in is en er zijn veel betere alternatieven beschikbaar. Bijvoorbeeld de open source graph database Neo4J. Met de volgende snippet maak je een bewandeling van een graaf.

```
TraversalDescription t = new TraversalDescription();
t.setOrder(TraversalDescription.DEPTH_FIRST);
t.setUniqueness(TraversalDescription.NODE);
t.setMaxDepth(2); t.setReturnFilter
(TraversalDescription.ALL);
```

Dat is een stuk compacter en beter leesbaar. Boven alles, is Neo4J veel efficiënter in het omgaan met grafen. Op moderne hardware is het mogelijk om tot een miljoen edges per seconde te bewandelen.

Ondanks de beschikbaarheid van betere alternatieven, laten we ons vaak verleiden tot het opslaan van grafen, bomen, gesorteerde lijsten en zelfs key-value paren in een relationele database. De voornaamste reden hiervoor is waarschijnlijk dat het brein van iedere programmeur ter wereld de laatste 25 jaar is geconditioneerd om te denken in SQL. Het is tijd om dat te doorbreken. Zeker nu veel van de NoSQL-alternatieven zich in productie hebben bewezen als stabiele, volwassen technologie.

Web scale?

Verschillende NoSQL-oplossingen worden geroemd om hun schaalbaarheid en performance. We kennen allemaal wellicht de verhalen van LinkedIn, Twitter of Facebook, waar clusters van honderden of duizenden machines worden ingezet om de gigantische hoeveelheid data en requests in realtime te verwerken. Maar wanneer spreken wij van een schalingsprobleem? Duizend transacties per seconde? Of tienduizend? Vijfhonderd duizend? En heb je daar een NoSQL-oplossing voor nodig? Dit zijn lastige vragen.

Het antwoord heeft over het algemeen te maken met een aantal factoren, waaronder: kosten (licenties, hardware, investering in ontwikkeling en kennisopbouw), verwachte groeicijfers van het aantal transacties en datavolume in de database. Er is geen gouden regel, maar een goede peiler is altijd de vraag of de totale belasting van de database door één enkele machine kan worden gedragen. Zowel nu als in de toekomst.

Als dat niet gaat, is een NoSQL-oplossing vaak een goede keuze. Relationele databases zijn van oudsher gemaakt en geoptimaliseerd om te draaien op één enkele machine en ook het relationele model en ACID transactieniveaus schalen lastig naar meerdere machines. Databases zoals bijvoorbeeld Apache Cassandra of Apache HBase zijn gebouwd met schaling naar meerdere machines als uitgangspunt. Dat heeft voor- en nadelen.

De voordelen:

- Schaalbaarheid is het uitgangspunt; lineair schalen is mogelijk.
- Door inzet van meerdere machines is hoge beschikbaarheid ook makkelijker te realiseren.
- Cluster oplossingen bieden hoge troughput rates.

De nadelen:

- Nieuwe technologie omarmen is altijd een investering in kennisopbouw, wat tijd kan kosten.
- Schaling met meerdere machines is complex en kan tijdrovende tuning met zich meebrengen.
- Veel (open source) oplossingen op dit gebied zijn (nog) niet zo volwassen als relationele databases.

Clusteroplossingen kunnen voordelen met zich meebrengen, maar onderschat ook niet wat er mogelijk is met een moderne machine met twee quad core CPUs, 256GB RAM, SSD hard drives en twee netwerkinterfaces. Hiermee kunnen we waarschijnlijk 90 procent van de problemen makkelijk aan.

Conclusie

NoSQL heeft absoluut een plaats in de huidige software-industrie, ook in enterprise-omgevingen. Daar waar schaling en datavolumes de pan uit reizen, maar ook daar waar we te maken hebben met een domein dat beter past bij een andere representatie dan een relationele database, maar bijvoorbeeld een grafendatabase of een document store.

Dat gezegd hebbende is het niet waarschijnlijk dat relationele databases of SQL uit het werkveld zullen verdwijnen. De database zoals we hem kennen zal zeker blijven bestaan, maar vaak worden aangevuld met NoSQL-oplossingen. «

In het volgende nummer van Java Magazine gaat Friso dieper in op een aantal aspecten van NoSQL.

Het denken in SQL van de laatste 25 jaar moet worden doorbroken.