integrace
e-business solutions

# RavenDB vs. SQL Server

Toepasbaarheid RavenDB
in Microsoft .Net ontwikkelshops

# 1. Introductie

RavenDB is een “open source 2nd generation document DB”. Net als vergelijkbare DBMS’en als MongoDB neemt het product radicaal afstand van het relationele model als basis voor opslagstructuren.

De door RavenDB gehanteerde manier van data-opslag biedt potentieel grote voordelen bij het opvragen van data, en bij het ontwikkelen van data-intensieve applicaties.

Het product is gebouwd in Microsoft .NET, en lijkt vooral voor .NET ontwikkelshops een prima alternatief voor SQL Server.

Voldoende reden om het product eens nader te bekijken dus. Dit document beschrijft RavenDB op hoofdlijnen, en vergelijkt het product met SQL Server.

Momenteel loopt binnen Integrace BV een praktijkevaluatie van RavenDB. De resultaten zullen we tezijnertijd publiceren als addendum op dit white paper.

# 2. Raven DB architectuur

## 2.1. Basisconcept

In RavenDB worden objecten en aggregaties daarvan opgeslagen in documenten. Als een dergelijk document wordt opgehaald is in één keer alle benodigde data beschikbaar voor de applicatie. Simpele 1-op-n-relaties (zoals: een nieuwsbericht heeft nul of meer tags) hoeven niet meer in aparte tabellen te worden opgeslagen en opgehaald. Daarnaast hoeft er weinig tot niks meer aan mapping te worden gedaan om objecten op een correcte wijze op te slaan. RavenDB converteert een compleet object inclusief alle deelverzamelingen in dat object automatisch naar JSON.

RavenDB heeft een behoorlijk goede pers, zeker onder ontwikkelaars. Waar bij SQL Server nogal wat tijd moet worden besteed aan performance-optimalisatie, is dat bij RavenDB veel minder een issue. Daarnaast vervalt met RavenDB een heel stuk werk rondom het vertalen van een objectenmodel naar een relationeel datamodel, en rondom het onderhouden van datalagen op dat relationele model.

## 2.2. NoSQL

De meeste Microsoft .NET ontwikkelshops werken met SQL Server, en hebben diepgaande kennis van de werking van relationele databasesystemen en SQL, de taal om data in en uit een dergelijke database te krijgen. RavenDB is echter niet relationeel, en de omschakeling van relationeel naar niet-relationeel, ook wel bekend onder de geuzennaam **NoSQL**, kan soms even slikken zijn.

SQL Server en RavenDB zijn beide databasesystemen, maar verschillen als dag en nacht. SQL Server vereist – uitzonderingen daargelaten – voor de meeste applicaties een verscheidenheid aan tabellen, één voor elk soort object dat wordt opgeslagen, en koppeltabellen voor de relaties tussen de onderlinge objecten. Een tabel kan per rij alleen een enkel plat object bevatten; voor het opslaan van meerdere tags bij een nieuwsbericht is een aparte tabel nodig. RavenDB heeft eigenlijk maar één tabel: de tabel met documenten. Omdat RavenDB 'schemaloos' is, kan elk soort object in een document worden opgeslagen.

Eén van de belangrijkste verschillen ligt in de manier waarop beide systemen met query's omgaan. In SQL Server kan een query enorm complex zijn, meerdere tabellen op verscheidene manieren aan elkaar koppelen en groeperingen en controles bevatten. Wanneer de query wordt uitgevoerd, analyseert SQL Server de query, formuleert een plan om de query uit te kunnen voeren, en voert de query vervolgens rechtstreeks op de data uit. RavenDB kan alleen simpele vergelijkingen op enkele documenten uitvoeren. Dat lijkt een tekortkoming, maar dat is het niet.

Een groot deel van het optimaliseren van query's in SQL Server vereist het van tevoren definiëren van indexen: opzoektabellen waardoor data sneller kan worden gevonden. Indexen zijn alleen van toepassing op een enkele tabel en een index beschrijft één of meer

kolommen die worden geïndexeerd. Maakt je query vergelijkingen met velden die niet in een index voorkomen? Dan ziet SQL Server geen andere optie dan sequentieel door de complete tabel te lopen om elk record te bekijken. Bij grote tabellen zorgt dit ervoor dat query's ontzettend traag worden.

RavenDB heeft een andere opzet. Wanneer je velden aanspreekt in een query, kijkt RavenDB eerst of er een index is die (ten minste) al die velden dekt. Is die er niet, dan wordt die ter plekke aangemaakt en wordt de query uitgevoerd met behulp van de nieuwe index. Een index is meestal van toepassing op documenten van een bepaald type, maar er kunnen ook indexen worden gemaakt die van toepassing zijn op alle documenten.

## 2.3. Eventually consistent

Hoe beide systemen het indexeren van nieuwe of aangepaste inhoud aanpakken is ook een belangrijk verschil. Wanneer je in SQL Server een record toevoegt of bijwerkt, zal eerst de inhoud in de tabel worden gezet. Daarna worden alle indexen op die tabel bijgewerkt. Pas daarna is de inhoud echt bijgewerkt. Dit betekent dat wanneer er een query wordt uitgevoerd, alle resultaten gegarandeerd nauwkeurig zijn. Dat principe is één van de onderdelen van het ACID-model.

RavenDB schrijft het document dat je hebt aangemaakt of bijgewerkt weg op schijf, en meldt daarna dat het voltooid is, waardoor je applicatie verder kan werken. Op de achtergrond, na het wegschrijven, worden vervolgens alle indexen nagelopen en waar relevant bijgewerkt. Het kan dus voorkomen dat de resultaten van een zoekopdracht niet helemaal accuraat zijn, omdat er een document is toegevoegd of aangepast, maar het bijwerken van de index die wordt aangesproken nog niet is voltooid. Dit principe staat bekend als 'eventually consistent'.

Het is belangrijk om op te merken dat hoewel het kan voorkomen dat indexen achterlopen op de actuele staat van de data—wat bekend staat als 'stale'—de inhoud van de documenten die worden opgehaald, of dat nou via een query of expliciet op naam is, wél altijd gegarandeerd accuraat is. Mocht het perse noodzakelijk zijn dat een zoekopdracht accurate resultaten geeft, dan kun je de verantwoordelijkheid voor het updaten in eigen handen nemen en je eigen mini-index maken in een apart document.

## 2.4. Snelheid

Omdat RavenDB bij het wegschrijven van documenten niet hoeft te wachten tot alle indexen zijn bijgewerkt, omdat RavenDB alleen vrij simpele vergelijkingen doet in query's, én omdat wat in SQL Server meerdere tabellen en query's kost in RavenDB in een enkel document en een enkele query kan, is RavenDB snel. RavenDB is niet in alle gevallen sneller dan SQL Server, maar wanneer query's complexer worden en tabellen groter komt RavenDB al snel op voorsprong te liggen.

## 2.5. Overige kenmerken

### 2.5.1. Lucene.Net

RavenDB maakt op de achtergrond gebruik van Lucene.Net om documenten te indexeren. Lucene.Net is de .NET-port van de open source-zoekindex Lucene. Een degelijke zoekindex met complexe features als facetten is dus ingebouwd.

### 2.5.2. Schaalbaarheid

Vanwege het 'eventually consistent' model kan RavenDB vrij simpel horizontaal schalen en dit is dan ook een ingebouwde feature. Iets wat met SQL Server onvermijdelijk tot hoofdbrekens en een lege portemonnee leidt.

### 2.5.3. Metadata

Documenten in RavenDB kunnen metadata bevatten. Metadata staat los van de inhoud van een document en bevat standaard informatie die RavenDB nodig heeft om documenten te kunnen materialiseren naar een object, evenals cache-gegevens. De metadata kan ook worden uitgebreid met applicatie-specifieke gegevens, zoals welke user een object heeft aangemaakt en autorisatiegegevens.

### 2.5.4. Uitbreidbaarheid

RavenDB is uitbreidbaar met plug-ins, bekend als 'bundles', waarmee extra functionaliteit kan worden geïntroduceerd. Een aantal voorbeelden zijn database-quota, automatisch verwijderen van documenten na een bepaalde tijd, versiebeheer en autorisatie.

Een belangrijke bundle is de zogeheten Index Replication-bundle. Hiermee kan de inhoud van een index worden gerepliceerd naar SQL Server. Erg handig, want één van de dingen waar RavenDB niet erg geschikt voor is, is rapportage. Veel scenario's kunnen worden opgelost met zogeheten map/reduce-indexen, maar voor bepaalde dingen heb je gewoon SQL Server nodig. Met de Index Replication-bundle blijft die mogelijkheid bestaan.

# 3. Vergelijking RavenDB met SQL Server

Onderstaande tabel vergelijkt RavenDB en SQL Server op hoofdpunten.

| **RavenDB** | **SQL Server** |
|---|---|
| Nieuw, relatief onbekend. | Oude gediende, bewezen technologie. |
| Hele objectgrafen kunnen in één keer worden gelezen en geschreven; beter geënt op OO-paradigma. | Stricte tabellen met tuples, wanneer 1-op-n of m-op-n relaties worden gemaakt moeten meerdere query's of 'dure' joins worden gemaakt. |
| Lucene query syntax; kan alleen waarde van eigenschappen testen. Groeperingen en berekende waardes kunnen met map/reduce indexen worden bereikt. In query's wordt nooit iets uitgerekend. | T-SQL; erg complexe querytaal, complexe en systeemslopende query's zijn mogelijk. |
| Leesquery's zijn (uitzonderingen daargelaten) **altijd** bloedsnel. | Snelheid van leesquery's is afhankelijk van de complexiteit, hoeveel en hoe goed de data geïndexeerd is. |
| Bijna 100% managed code; buffer overflows en dergelijke vulnerabilities onmogelijk. | 100% native code; buffer overflow en dergelijke mogelijk. |
| Schemaloos; maakt aanpassen van datamodel ontzettend makkelijk. | Datamodel gebaseerd op schema's; aanpassingen kunnen soms complete herbouw van tabellen vereisen. |
| Indexen kunnen zowel handmatig worden gedefinieerd als automatisch. | Indexen kunnen alleen handmatig worden gedefinieerd. |
| Bedrijf is vrij klein, continuïteit is niet gegarandeerd. | Microsoft zal niet zo maar ten onder gaan. |
| Bedrijf is vrij klein, supportlijnen zijn erg kort. | Support beschikbaar, maar complexe support-structuur. |
| Schrijfsnelheid alleen afhankelijk van I/O en in sommige gevallen triggers. | Schrijfsnelheid is erg onvoorspelbaar; afhankelijk van hoeveelheid indexen, welke indexen, wat voor veldtypes er gebruikt worden in indexen, triggers, I/O, etc. |
| Wegschrijven is ACID (Atomic, Consistent, | Wegschrijven is ACID, lezen kan ook ACID |

| | |
|---|---|
| Isolated, Durable). | zijn. |
| Indexeren gebeurt altijd asynchroon, wegschrijven is klaar zodra document is weggeschreven. | Indexeren gebeurt altijd synchroon, wegschrijven is pas klaar nadat alle rijen zijn weggeschreven én geïndexeerd. |
| Indexen zijn BASE (Basically available, Soft state, Eventual consistency); stale state kan voorkomen. | Indexen zijn ACID; stale data kan met de juiste transactie-isolatie niet voorkomen. |
| $25/maand/instance | Veel kostbaarder; met recente wijziging prijsmodel op basis van core's ipv cpu's nog duurder geworden voor omvangrijke installaties |
| REST-API of LINQ-API. | Eindeloze hoeveelheid API's die allemaal SQL uitvoeren. |
| Makkelijk uitbreidbaar met extra functionaliteit. | Niet uitbreidbaar, behalve door MS. |
| Documentatie nog niet altijd volledig of up-to-date. | Uitgebreide documentatie. |
| Horizontaal schalen gratis en ingebouwd | Horizontaal schalen moeilijk en duur. |
| Ad-hoc query's af en toe lastig (makkelijker met LINQPad driver). | Ad-hoc query's geen probleem. |
| Kan volledig en full-featured door applicatie zelf gehost worden; oplossing indien systeembeheer huiverig is voor onbekende applicaties. | Kan stand-alone draaien, maar met erg veel beperkingen. |
| Full-text search ingebouwd en gebaseerd op Lucene. | Full-text search ingebouwd, maar erg beperkt en lastig om mee te werken. |
| Documenten kunnen meta-data bevatten, die niet in de data zelf te zien zijn. | Alle meta-data dient in het datamodel te zitten. |
| Versioning ingebouwd. | Voor versioning dient het datamodel radicaal te worden aangepast, het bijhouden van versies is ingewikkeld. |
| Autorisatie ingebouwd, transparant en krachtig | Geen enkele autorisatie ingebouwd, moet door applicatie zelf worden geregeld. |
| Door middel van een bundle kunnen quota worden afgedwongen | Quota zijn makkelijk instelbaar. |

| RavenDB Studio | SQL Server Management Studio |
| --- | --- |
| Zo goed als geen systeembeheer noodzakelijk (met uitzondering van back-ups), zelfoptimaliserend. | Maar voor een deel zelfoptimaliserend, veel systeembeheer noodzakelijk. |
| Open source; iedereen kan fouten vinden, maar ook fixen | Closed source; niemand kan problemen verhelpen behalve MS |

# 4. Conclusies & Aanbevelingen

## 4.1. Toepassingsdomein

De keuze van DBMS moet in eerste instantie worden gebaseerd op de soort toepassing. RavenDB is uitermate geschikt voor zogeheten read-heavy applicaties. Vanwege achtergrondindexering en de ondersteuning voor map/reduce-indexen biedt het DBMS ook voordelen bij write-heavy applicaties. Ook applicaties met een dermate grote belasting dat verticaal schalen onpraktisch is of waar horizontaal schalen moeilijk of duur is kunnen profiteren van RavenDB. Applicaties die voornamelijk leunen op rapportage zijn minder geschikt voor RavenDB.

## 4.2. Ontwikkelsnelheid

RavenDB kan leiden tot een significante versnelling van ontwikkeltrajecten. Een fors deel van het ontwikkelwerk vervalt simpelweg: er hoeft geen databasemodel meer worden gemaakt en daarmee vervalt een groot deel van de datalaag ook. Omdat RavenDB als DBMS redelijk zelf-onderhoudend is, valt ook een groot deel systeembeheer weg. Wat overblijft, is tijd om goed na te denken over het objectmodel, wat je in een document gaat opslaan en wat de zogeheten ‘transaction boundary’ is - welke informatie gaat worden aangepast in een enkele transactie.

## 4.3. Performance

Mits correct toegepast zal de performance en schaalbaarheid van applicaties die RavenDB gebruiken een stuk hoger liggen dan applicaties die SQL Server gebruiken. Hybride gebruik is ook mogelijk, waarbij SQL Server als opslag wordt gebruikt en RavenDB als cache-laag.

## 4.4. Hybride toepassingen

RavenDB lijkt al met al klaar voor een bredere toepassing. De belangrijkste barrière voor een bredere inzet van RavenDB lijkt de relatieve onbekendheid en nog geringe verspreiding van deze technologie.

Om de onzekerheden die hierdoor ontstaan weg te nemen is het verstandig om in applicatie-architecturen te zorgen voor een zo gering mogelijke afhankelijkheid van het product, zodat als de nood onverhoopt toch aan de man komt eenvoudig kan worden teruggegaan naar SQL Server.