



MySQL talk

Trage website?

Het optimaliseren van een bestaande website die een MySQL database heeft is niet altijd even makkelijk. Het probleem kan namelijk op veel verschillende plekken zitten: de database layout, de queries, de php code zelf, een externe service, javascript, ... Vandaag ga ik uiteraard vertellen over het database optimalisatie deel.

't Is de database!

Vaak situeert het probleem zich in de database of is de manier om de data op te halen niet altijd even goed in orde. Op zich is het 'goed' wanneer de website traag is doordat de database of de queries niet in orde zijn, dat betekent dat je door een paar optimalisaties aan te brengen aan de database, de website al sneller laat lopen. Dus je moet geen ellenlange code aanpassen (al dan niet door iemand anders geschreven) en loopt dus veel minder kans om fouten te creëren. Al bij al is het dus een goede zaak wanneer het probleem zich op database niveau situeert.

Eerste manier van optimaliseren

De allereerste manier van optimaliseren is vaak niet mogelijk, maar voor de volledigheid geef ik het even mee: upgrade de MySQL server installatie van 4.x naar 5.x. Ik heb zelf persoonlijk ondervonden dat de aanpassingen van 4 naar 5 een grote stap voorwaarts zijn. Zo heb ik bij het testen van een paar voorbeeld queries voor jullie niets meer te optimaliseren, ze draaiden al tegen volle snelheid. Het is dus vaak de moeite om de software op de server up-to-date te houden, voor zowel security als snelheid. Maar helaas is dit zoals gezegd vaak niet mogelijk.

Tweede manier van optimaliseren

In MySQL 4 was de plaatsing van je condities soms zeer belangrijk en moest je alles zo narrow mogelijk maken. Ondanks de grote verbeteringen van 4 naar 5 blijft het toch belangrijk om hier over na te denken. Zo kun je door wat na te denken het aantal resultaten waar rekening mee gehouden moet worden flink verminderen. Natuurlijk zal het vaak niet uitmaken hoe je je queries gaat opbouwen, het zullen vooral de indexen zijn die de grote verbeteringen gaan opleveren, maar op een gegeven punt kun je niet meer indexen leggen en zul je toch de queries moeten optimaliseren. Daarom is het dus toch vrij belangrijk om dit te doen.

Voorbeeld:

We moeten de accommodaties selecteren die 3 sterren hebben, en in land met ID 10 zitten.

```
SELECT
  a.id, a.name, a.price, a.image_url, c.id AS country_id, c.name AS country_name, r.id
AS region_id, r.name AS region_name, ci.id AS city_id, ci.name AS city_name
```

```

FROM
  accomodations a
  INNER JOIN
    cities ci
    ON
      ci.id = a.city_id
  INNER JOIN
    regions r
    ON
      r.id = a.region_id
  INNER JOIN
    countries c
    ON
      c.id = a.country_id
  INNER JOIN
    accomodations_info ai
    ON
      ai.accomodation_id = a.id
WHERE
  ai.name = 'stars'
AND
  ai.value = '3'
AND
  c.id = 10

```

Deze query ziet er op zich niet zo spectaculair uit, en dat is ook zo. Maar om iets te verbeteren moet je niet gewoon deze query bekijken, maar ontdekken wat er gaat gebeuren bij het uitvoeren ervan. Dit kun je zeer simpel doen door het EXPLAIN statement te gebruiken. De output daarvan zal je de nodige informatie geven die we nu gaan gebruiken.

En dit is de output van het EXPLAIN statement op onze query:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	c	const	PRIMARY	PRIMARY	4	const	1	
1	SIMPLE	a	ref	PRIMARY,country_id,region_id,city_id	country_id	5	const	1714	Using where
1	SIMPLE	ci	eq_ref	PRIMARY	PRIMARY	4	zomerbrochure.a.city_id	1	
1	SIMPLE	r	eq_ref	PRIMARY	PRIMARY	4	zomerbrochure.a.region_id	1	
1	SIMPLE	ai	ref	accomodation_id,name_2,name	accomodation_id	4	zomerbrochure.a.id	3	Using where

Wat meer uitleg over deze output. Allereerst zie je het select type. In dit geval hebben we te doen met SIMPLE queries. Er zijn er nog vele andere bv. UNION, SUBQUERY, DERIVED, etc. Op zich zul je in 90% van de gevallen met een SIMPLE query te maken hebben. Het volgende interessante is het type. Ook hier zijn er vele soorten: const, eq_ref, ref, fulltext, range, etc... De meest voorkomende types zijn ook diegene die in dit voorbeeld staan, vandaar dat ik even wat meer uitleg ga geven over deze 3 types en nog een paar andere:

- **Const:** Dit wil zeggen dat er maar 1 rij gematcht zal worden, vaak omdat er een conditie op de primary key gezet zal worden. Dit type is dus ongelooflijk snel
- **Eq_ref:** Dit is het 2de snelste type. Dit wil zeggen dat er slechts 1 andere rij zal gematcht worden bij een rij, dit type duidt dus een 1 op 1 relatie.
- **Ref:** Deze is niet meer zo snel als de vorige, maar het kan slechter. Dit wil zeggen dat er voor geen 1 op 1 relatie meer is, maar dat er toch slechts een bepaald aantal resultaten gematcht zullen worden.
- **INDEX:** Dit is het op 1 na slechtste type, deze join moet de volledige index afgaan om een resultaat te bekomen.
- **ALL:** Dit is het allerslechtste wat er kan zijn. Deze join wil zeggen dat er een full table scan moet gebeuren en dus zelfs geen index gebruikt zal worden. Wanneer

je dit ziet is er werk aan de winkel en moet je proberen om ten minste een index te leggen.

De laatste 2 zijn niet optimaal, maar vooral het ALL type is zeker en vast te vermijden. De rest van de kolommen van de output spreken voor zich denk ik.

Om verder te gaan met de query:

Zoals je ziet worden hier tot 1714 rijen gematcht bij de accomodations. Op zich is dit al heel goed, maar het kan veel beter, met slechts een minieme, maar toch logische aanpassing.

Dit is de nieuwe, aangepaste query:

```
SELECT
  a.id, a.name, a.price, a.image_url, c.id AS country_id, c.name AS country_name, r.id
AS region_id, r.name AS region_name, ci.id AS city_id, ci.name AS city_name
FROM
  accomodations a
  INNER JOIN
    cities ci
    ON
      ci.id = a.city_id
  INNER JOIN
    regions r
    ON
      r.id = a.region_id
  INNER JOIN
    countries c
    ON
      c.id = a.country_id
  INNER JOIN
    accomodations_info ai
    ON
      ai.accomodation_id = a.id
WHERE
  ai.name = 'stars'
AND
  ai.value = '3'
AND
  ci.country_id = 10
```

Er is dus enkel een verandering op de join, in plaats van hem op het land te leggen, leggen we hem op de stad.

En dit is de output van het EXPLAIN statement:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	ci	ref	PRIMARY,country_id	country_id	5	const	463	Using where
1	SIMPLE	a	ref	PRIMARY,country_id,region_id,city_id	city_id	5	zomerbrochure.ci.id	5	Using where
1	SIMPLE	r	eq_ref	PRIMARY	PRIMARY	4	zomerbrochure.a.region_id	1	
1	SIMPLE	c	eq_ref	PRIMARY	PRIMARY	4	zomerbrochure.a.country_id	1	
1	SIMPLE	ai	ref	accomodation_id,name_2,name	accomodation_id	4	zomerbrochure.a.id	3	Using where

Zoals je ziet worden hier maar maximum 463 rijen overwogen, in plaats van de 1714. Op zich is dit niet zo'n grote verbetering, maar stel je voor dat er meer data in de database komt te staan, dan moet de eerste query steeds meer dan 4x meer werk doen dan de 2de. Dit maakt het feit dat het type van de join (Const => Ref) er op achteruit gaat toch een de moeite.

Bij een kleine resultset is dit eerlijk gezegd vaak te verwaarlozen, maar het maakt wel degelijk uit.

Hoe beslis je welke velden je gaat matchen? Wel, vaak is het een kwestie van logisch na te denken. Er zijn meer steden dan landen, en elke stad heeft een land. In de eerste query gaan we eerst een join leggen van alle accommodaties op de steden, daarna op de regio's en daarna op het land. Dit komt er op neer dat eerst alle steden moeten gelinkt worden aan alle accommodaties, etc... Door de conditie op stad niveau te leggen gaan we er niet meer van uit dat alle steden in acht genomen worden, maar enkel de steden met het correcte land. Als je nadenkt over hoeveel steden er ter wereld zijn, snap je wel dat dit een gigantisch verschil kan zijn. Misschien dat sommigen onder jullie aan het denken zijn 'Zou de country_id conditie niet beter thuishoren op de accommodations tabel?'. Nee, simpelweg omdat er meer data in die tabel zit. Ook al zet je een index op country_id, deze index is sowieso groter, en het overlopen van die index zal zeker en vast meer tijd vragen dan het overlopen van de index op de country_id gegevens van de stad wanneer je met een propvolle database te maken hebt. Hoe kun je dit allemaal weten? Je moet gewoon de database een beetje kennen. Een korte analyse is zeker en vast nodig indien je als buitenstaander een query moet optimaliseren, maar als je de database zelf ontwikkeld hebt zal dit vaak vanzelf komen.

Derde manier van optimaliseren

De volgende manier van optimalisatie is ook een eenvoudige, maar toch vaak vergeten stap: indexen leggen. We nemen even onze bekende query:

```
SELECT
  a.id, a.name, a.price, a.image_url, c.id AS country_id, c.name AS country_name, r.id
  AS region_id, r.name AS region_name, ci.id AS city_id, ci.name AS city_name
FROM
  accommodations a
  INNER JOIN
    cities ci
    ON
      ci.id = a.city_id
  INNER JOIN
    regions r
    ON
      r.id = a.region_id
  INNER JOIN
    countries c
    ON
      c.id = a.country_id
  INNER JOIN
    accommodations_info ai
    ON
      ai.accomodation_id = a.id
WHERE
  ai.name = 'stars'
AND
  ai.value = '3'
AND
  ci.country_id = 10
```

Zoals je ziet zijn er redelijk wat joins en condities. Als we gewoon de standaard database opstellen leggen we vaak wel een primary key vast, wat automatisch een PK index tot

gevolg heeft. Maar vaak is dit niet voldoende. Als we in het geval van deze query geen extra indexen leggen zullen er zeker en vast problemen ontstaan qua performance. Vaak weten we niet goed hoe we dit probleem moeten aanpakken, maar het komt gewoon neer op even goed en logisch nadenken.

Stel dat de database uit dit voorbeeld een gewone standaard database is, met een PK index en niets anders en we gaan indexen leggen dan beginnen we bij het begin:

- FK's:

Foreign keys zullen vrij vaak gebruikt worden, deze zitten namelijk steeds in de join voorwaarde. Het is daarom zeer aan te raden om een index te leggen op de velden:

- a.city_id
- a.region_id
- a.country_id
- ai.accomodation_id
- ci.country_id

Op deze manier zullen de join condities al veel sneller verlopen en zal de load van de database server zakken.

- Conditioes (WHERE):

Dit is een moeilijker punt. Vaak leggen we op allerhande plekken condities op en zijn we geneigd om toch maar een extra index op veld X te leggen, want dat zal toch wel ergens gebruikt worden. Dit is echter géén goede denkpiste! Een index zou namelijk in het geheugen moeten blijven zitten om snel te zijn. Ga je echter nutteloze indexen leggen, zal het geheugen snel vol raken en zal de server de indexen gaan bewaren op de schijf, wat uiteraard veel trager zal zijn. Het is daarom belangrijk om enkel te indexen als het uitmaakt. In het voorbeeld van deze query leggen we een index op:

- ai.name
- ai.value

Mag dit zomaar een index zijn? Nee, je moet weten om welk soort veld het gaat, in dit geval zijn het 'TEXT' velden, deze moeten dus een fulltext index krijgen om te kunnen werken. Op een veld met een fixed size (int, varchar, etc..) kun je wel een gewone index leggen.

Dankzij deze indexen zal de query veel sneller uitgevoerd worden, de load zakken en is er veel minder LOCK tijd op de tabellen. Allemaal zeer goede zaken.

Slot

Hopelijk was deze MySQL talk interessant en heeft hij jullie wat bijgebracht! Hoewel het enorm simpele zaken zijn, vergeten we deze dingen heel vaak. Ik hoop dat deze simpele talk toch de moeite waard was.

Vragen?