Postgress: een relationele database

Steven Raaijmakers

February 2017

1 Inleiding

Tegenwoordig maken veel technologische diensten gebruik van databases. Deze diensten gebruiken de databases om data in op te slaan. Hierbij is de kunst deze data gestructureerd op te slaan, op een flexibele manier. Eveneens moet de data gemakkelijk geraadpleegd worden door de gebruikers.

Er bestaan veel verschillende types database, waarvan de relationele database er één is. PostgreSQL is een databaseserver die een relationele database runt.

2 Theorie

2.1 Relationele database

Een relationele database is een database bestaande uit verschillende tabellen die naar elkaar kunnen linken via een unieke identifiers.

Een tabel kan bijvoorbeed een kolom in zijn rij hebben die linkt naar de key van een rij uit een andere tabel.

Denk bijvoorbeeld aan een muziek-database, met daarin artiesten en nummers. Een artiest maakt vaak meerdere nummers, en kan naast zijn naam bijvoorbeeld ook zijn geboortedatum bevatten. Daarom heeft een artiest een unieke key (een getal). Elke row in de nummers tabel bevat een nummer, en in een dergelijke rij bevindt zich de kolom "artiest". Deze kolom wordt dan dus gevuld met de identifier van de juiste artiest, in plaats van enkel zijn naam.

2.2 Locking

Bij het bewerken van een database kunnen verschillende transacties zoals het lezen of schrijven van data-elementen zich parallel afspelen.

Als er meerdere transacties parallel plaatsvinden is het volgens het ACID principe belangrijk dat een data element alleen verandert kan worden wanneer geen enkele andere transactie op dat moment hetzelfde data element uitleest. Wel is er de mogelijkheid om verschillende transacties hetzelfde data-element gelijktijdig te laten bekijken.

De database community heeft hierop het *locken* bedacht. In Postgres zijn er verschillende soorten locks waarvan de *shared-lock* en de *exclusive-lock* het belangrijkst zijn. De shared-lock wordt toegepast op data elementen die enkel gelezen worden. De exclusive-lock wordt toegepast wanneer een data-element geschreven moet worden. De exclusive-lock (en daarmee de write) worden pas toegestaan als er op data-element geen *read*s meer plaatsvinden. Hiermee wordt de consistentie van de data-elementen gegarandeerd.

3 Experimenten

3.1 Locking

Schedule A $r_1(A); r_2(B); r_3(C); w_1(B); w_2(C); w_3(A)$

T1	T2	T3
read(A)		
	read(B)	
		read(C)
write(B)		
	write(C)	
		write(A)

	T1	T2	T3
	lock-S(A) read(A)		
		lock-S(B) read(B)	
			lock-S(C) read(C)
i	**lock-X(B)** DENIED		
		lock-X(C) DENIED	
			lock-X(A) DENIED
	unlock(A)	unlock(B)	unlock(C)

ii A, B en C worden achter elkaar uitgelezen. Omdat de *unlocks* pas op het einde van de transactie staan volgens TLP, ontstaan er problemen bij de eerste lock-X(B) bij T1. Er ontstaat een zogeheten deadlock, omdat de lock-S(B) nog openstaat in T2. Hetzelfde geldt voor de exclusive locks van C en A. Er ontstaat een deadlock

iii Zie schedule bij i.

iv Er zijn is geen upgrade strategie mogelijk bij dit schedule omdat er een deadlock in zit. Voor een upgrade zou dezelfde transactie namelijk al een shared lock voor B moeten hebben wanneer de write wordt aangevraagd, en dus de read wordt geupgrade. Deze staat echter in een andere transactie waardoor er een normale exclusive lock wordt aangevraagd wat tot dezelfde situatie leidt als bij i.

T1	T2	T3
lock-U(*A*)		
read(A)		
	lock-U(*B*)	
	read(B)	
		lock-U(*C*)
		read(C)
lock-X(*B*)		
write(B)		
	lock-X(*C*)	
	write(C)	
		lock-X(*A*)
		write(A)
unlock(A, B)	unlock(B, C)	unlock(C, A)

v

vi Eerst zullen A, B en C worden gelezen waarna B, C en A geschreven worden

Schedule B $r_1(A); r_2(B); r_3(C); r_1(B); r_2(C); r_3(D); w_1(C); w_2(D); w_3(E);$

T1	T2	T3
read(A)		
	read(B)	
		read(C)
read(B)		
	read(C)	
		read(D)
write(C)		
	write(D)	
		write(E)

i

T1	T2	T3
lock-S(A) read(A)		
	lock-S(B) read(B)	
		lock-S(C) read(C)
lock-S(B) read(B)		
	lock-S(C) read(C)	
		lock-S(D) read(D)
lock-X(C) DENIED		
	lock-X(D) DENIED	
		lock-X(E) write(E)
unlock(A, B)	unlock(B, C)	unlock(C, D, E)

ii Enkel E zal geschreven worden omdat deze bij de write nog geen openstaande locks heeft in tegenstelling tot de C en D, die dus niet geschreven zullen worden.

iii Zie i

iv De scheduler zal het schema op dezelfde manier uitvoeren als besproken in ii, aangezien de upgrades enkel gelden wanneer er van een shared lock naar een exclusive lock gegaan wordt binnen dezelfde transactie. Dit is bij alle 3 de transacties niet het geval.

	T1	T2	T3
	lock-S(*A*)		
	read(A)		
		lock-S(*B*)	
		read(B)	
			lock-U(*C*)
			read(C)
	lock-S(*B*)		
	read(B)		
		lock-S(*C*)	
v		read(C)	
			lock-U(*D*)
			read(D)
	lock-X(*C*)		
	write(C)		
		lock-X(*D*)	
		write(D)	
			lock-X(*E*)
			write(E)
	unlock(A, B, C)	unlock(B, C, D)	unlock(C, D, E)

vi De scheduler zal A, B en C achter elkaar uitlezen. Vervolgens B, C en D. Daarna worden C, D en E beschreven en in de database terug gestopt.

Schedule C
$r_1(A); r_2(B); r_3(C); r_1(B); r_2(C); r_3(A); w_1(A); w_2(B); w_3(C);$

T1	**T2**	**T3**
read(A)		
	read(B)	
		read(C)
read(B)		
	read(C)	
		read(A)
write(A)		
	write(B)	
		write(C)

i

T1	**T2**	**T3**
lock-X(*A*)		
read(A)		
	lock-X(*B*)	
	read(B)	
		lock-X(*C*)
		read(C)
lock-S(*B*)		
DENIED		
	lock-S(*C*)	
	DENIED	
		lock-S(*A*)
		DENIED
lock-X(*A*)		
DENIED		
	lock-X(*B*)	
	DENIED	
		lock-X(*C*)
		DENIED
unlock(A)	unlock(B)	unlock(C)

ii Veel van de locks zullen denied worden omdat de exclusive locks van die data elementen op dat moment nog openstaan. Bij uitvoer van dit schedule zal dus enkel A, B en C uitgelezen worden en niet bewerkt.

iii

T1	T2	T3
lock-S(A)		
read(A)		
	lock-S(B)	
	read(B)	
		lock-S(C)
		read(C)
lock-S(B)		
read(B)		
	lock-S(C)	
	read(C)	
		lock-S(A)
		read(A)
lock-X(A)		
write(A)		
	lock-X(B)	
	write(B)	
		lock-X(C)
		write(C)
unlock(A, B)	unlock(B, C)	unlock(C, A)

iv Het schema zal worden uitgevoerd naar behoren omdat het upgraden bij dit schedule mogelijk is. Dit komt doordat de geschreven elementen eerder in dezelfde transactie een shared lock toegewezen krijgen. Eerst wordt A, B en C gelezen. Daarna wordt B, C en D gelezen waarna A, B en C geschreven worden.

v

T1	T2	T3
lock-S(A)		
read(A)		
	lock-S(B)	
	read(B)	
		lock-S(C)
		read(C)
lock-U(B)		
read(B)		
	lock-U(C)	
	read(C)	
		lock-U(A)
		read(A)
lock-X(A)		
DENIED		
	lock-X(B)	
	DENIED	
		lock-X(C)
		DENIED
unlock(A, B)	unlock(B, C)	unlock(C, A)

vi Ook in deze scheduler zal het schema uitgevoerd worden zoals verwacht. De data-elementen worden eerst uitgelezen waarna ze (in een andere transactie) weer geschreven worden.

Schedule D
$r_1(A); r_2(B); r_3(C); w_1(B); w_2(C); w_3(D);$

T1	T2	T3
read(A)		
	read(B)	
		read(C)
write(B)		
	write(C)	
		write(D)

---

i

T1	T2	T3
lock-S(A)		
read(A)		
	lock-S(B)	
	read(B)	
		lock-S(C)
		read(C)
lock-X(B)		
DENIED		
	lock-X(C)	
	DENIED	
		lock-X(D)
		write(D)
unlock(A)	unlock(B)	unlock(C, D)

ii Alleen D zal geschreven worden omdat B en C nog een shared lock hebben op het moment dat ze geschreven worden.

iii

T1	T2	T3
lock-S(A)		
read(A)		
	lock-S(B)	
	read(B)	
		lock-S(C)
		read(C)
lock-X(B)		
write(B)		
	lock-X(C)	
	write(C)	
		lock-X(D)
		write(D)
unlock(A, B)	unlock(B, C)	unlock(C, D)

iv A, B, C worden gelezen. Daarna wordt B, C, D geschreven.

T1	T2	T3
lock-S(A)		
read(A)		
	lock-U(B)	
	read(B)	
		lock-U(C)
		read(C)
lock-X(B)		
write(B)		
	lock-X(C)	
	write(C)	
		lock-X(D)
		write(D)
unlock(A, B)	unlock(B, C)	unlock(C, D)

v

vi Zelfde resultaat als bij iv.

Schedule E $r_1(A); r_2(B); r_3(C); r_1(B); r_2(C); r_3(D); w_1(A); w_2(B); w_3(C);$

T1	T2	T3
read(A)		
	read(B)	
		read(C)
read(B)		
	read(C)	
		read(D)
write(A)		
	write(B)	
		write(C)

i

T1	T2	T3
lock-X(*A*)		
read(A)		
	lock-X(*B*)	
	read(B)	
		lock-X(*C*)
		read(C)
lock-S(*B*)		
DENIED		
	lock-S(*C*)	
	DENIED	
		lock-S(*D*)
		read(D)
lock-X(*A*)		
DENIED		
	lock-X(*B*)	
	DENIED	
		lock-X(*D*)
		DENIED
unlock(A)	unlock(B)	unlock(C, D)

ii A, B en C worden gelezen. Daarna worden er locks aangevraagd die DENIED worden omdat die data-elementen hun exclusive lock nog hebben openstaan. Hierna wordt D uitgelezen. Vervolgens kunnen de writes niet plaatsvinden omdat de exclusive locks van dezelfde elementen nog openstaan.

iii

T1	T2	T3
lock-S(A)		
read(A)		
	lock-S(B)	
	read(B)	
		lock-S(C)
		read(C)
lock-S(B)		
read(B)		
	lock-S(C)	
	read(C)	
		lock-S(D)
		read(D)
lock-X(A)		
write(A)		
	lock-X(B)	
	write(B)	
		lock-X(C)
		write(C)
unlock(A, B)	unlock(B, C)	unlock(C, D)

iv De scheduler zal eerst A, B en C uitlezen. Daarna leest de scheduler B, C, D uit waarna hij zonder problemen A, B en C zal schrijven.

v

T1	T2	T3
lock-U(A)		
read(A)		
	lock-S(B)	
	read(B)	
		lock-S(C)
		read(C)
lock-U(B)		
read(B)		
	lock-U(C)	
	read(C)	
		lock-S(D)
		read(D)
lock-X(A)		
write(A)		
	lock-X(B)	
	write(B)	
		lock-X(C)
		write(C)
unlock(A, B)	unlock(B, C)	unlock(C, D)

vi Zie iv.

3.2 Musicbrainz opstarten

Bij het installeren van Postgress op ubuntu-systemen wordt er automatisch de user "postgress" aangemaakt. Deze beschikt over alle rechten die nodig zijn voor de postgress database. Op deze user kan ingelogd worden volgens:

```
sudo su postgres
```

Na het installeren van de musicbrainz-database (via: `https://bitbucket.org/lalinsky/mbslave`) kan deze geladen:

```
psql musicbrainz
```

Door het *searchpath* aan te passen wordt het overbodig om bij elk commando aan te geven dat het deze database betreft:

```
SET search_path TO musicbrainz;
```

Omdat we werken met tabellen die rows hebben die breeder zijn dat het scherm is het handig om de "Expanded Display"-optie aan te zetten. Hierdoor worden de rijen van een tabel overzichtelijker weergegeven:

```
\x
```

3.3 Musicbrainz bewerken

In het schema van `https://wiki.musicbrainz.org/-/images/5/52/ngs.png` is te zien uit welke tabellen en relaties de Musicbrainz is aangepast.

3.3.1 Toevoegen

In dit overzicht zien we dat een tabel genaamd *artist_type*. Wanneer men een rij wil toevoegen aan deze tabel met bepaalde waardes, is het handig om eerst meer informatie op te vragen over deze specifieke tabel. Dit kan via:

```
\d musicbrainz.artist_type
```

We zien dat de tabel uit meerdere kolommen bestaat, waarvan de velden id, name, *child_order* en gid bij een nieuwe toevoeging NOT NULL mogen zijn. Tevens valt te zien dat id en *child_order* wel een DEFAULT waarde hebben. De invulling van deze waarde kunnen we dus overlaten aan de database zelf. Dit rest nog twee velden die zelf ingevuld moeten worden: name en gid. Het gid velt betreft een UUID, en omdat postgres niet in staat is zelf deze waarde in te vullen genereren we zelf een UUID. Hierdoor ontstaat de volgende query waarmee we het artist-type "duo" toevoegen aan de *artist_type* tabel in musicbrainz.

```
INSERT INTO artist_type (name, gid)
VALUES ('Duo', '8acdb6ef-6946-4de8-9cf1-e93700c9c249');
```

3.3.2 Verwijderen

Bij het verwijderen van een row uit de artist tabel is het handig om te verwijderen aan de hand van een ID, aangezien elke artiest een unieke ID heeft.

```
DELETE FROM artist WHERE id='1228042'
```

3.3.3 Tabellen linken

De musicbrainz database is een relationele database. Meerdere tabellen aan elkaar linken op basis van keys kan via de INNER JOIN. De volgende query kijkt in de release tabel waarin zich een "artist_credit"-veld bevindt dat linkt naar een row uit de "artist_credit"-tabel.

```
SELECT release.name AS release, artist_credit.name as artist
FROM release
INNER JOIN artist_credit on
release.artist_credit = artist_credit.id
ORDER BY artist_Credit.name
LIMIT 50;
```

4 Resultaat

4.1 Musicbrainz bewerken

4.1.1 Toevoegen

De eerder toegevoegde waarde "duo"aan de *artist_type* tabel kan worden opgevraagd door middel van de volgende query:

```
SELECT * FROM artist_type ORDER BY id DESC
```

We zien hierin dat de waarde juist is toegevoegd aan de tabel.

4.1.2 Verwijderen

De verwijderde artiest kunnen we proberen terug te zoeken door hem te selecteren aan de hand van zijn ID:

```
SELECT * FROM artist WHERE id='1228042'
```

Omdat het resultaat 0 rows oplevert kunnen we er vanuit gaan dat de artiest correct verwijdert is.

5 Discussie en Conclussie

Postgres is een goede optie wanneer je gebruik wil maken van een relationele database. Relationele databases zijn eenvoudig te snappen voor de gebruikers met enige technische kennis. Het schrijven van de queries is lastig wanneer je geen visuele weergave hebt van de database waarin je werkt. Daarom is het verstandig een interface met een GUI te gebruiken over Postgres heen. Hierdoor hoef je niet telkens metadata van de database op te vragen. Ook is het fijn om een visuele voorstelling te hebben waardoor je direct kan zien hoe de database bewerkt wordt op bepaalde commando's.

Ook is te zien dat het locken beter kan worden overgelaten aan de lock-manager, omdat het moeilijk is de juiste lock soorten te onthouden met de bijbehorende compatibility.