SQL assertions

Onbekend, dus onbemind?

Weet u wat SQL assertions zijn? Als u meent een database professional te zijn, zou dat het geval moeten zijn. En gelijksoortige vragen zouden kunnen zijn: doet u nog aan datamodellering? Maakt u nog nette relationele database designs? Weet u wat 3NF is? Of hoe u om moet gaan met redundante gegevens? Als u database applicaties maakt, zou u hier antwoord op moeten kunnen geven. Helemaal als u Oracle met de huidige SQL optimizer als fundament van die applicatie gebruikt. Ik wil in dit artikel SQL assertions behandelen.

SQL assertions zijn feitelijk constraints. Ze bieden je de mogelijkheid om tegen het DBMS te zeggen dat een bepaalde boolean expressie bij elke database toestand naar TRUE moet evalueren. Bijvoorbeeld: we hebben een EMP (employee) tabel in ons database design, en de volgende business rule (beter gezegd: constraint) dient geïmplementeerd te worden.

Er mag hooguit één PRESIDENT zijn

Met een assertion wordt deze constraint als volgt vastgelegd:

```
create assertion EEN_PRESIDENT as
check( (select count(*) from EMP where JOB = 'PRESIDENT') <= 1));
```

De expressie tussen de ronde haken van het CHECK keyword is van type boolean. Het significante verschil met de CHECK clausule die we al kennen bij het CREATE TABLE statement, is dat de boolean expressie van een assertion wel queries kan bevatten. Een assertion is hierdoor het vehikel binnen de ANSI SQL standaard om alle andere data constraints declaratief kenbaar te maken aan het DBMS.
Door een constraint te declareren hoeven we zelf niet meer werk te besteden aan het procedureel implementeren van die constraint: constraint implementatie, ofwel zorgen dat de boolean expressie door geen enkele transactie geschonden wordt, is bij een assertion de schone taak van het DBMS geworden. Voordat u nu denkt: 'Wat heb ik gemist. Wanneer is deze feature beschikbaar gekomen?', moet ik meteen al melden dat support voor SQL assertions nog steeds niet aanwezig is in de huidige DBMS-en. Desalniettemin dient u als database professional wel bekend te zijn met het concept. Al is het alleen maar om database leveranciers erop te attenderen dat ze assertions eindelijk eens moeten gaan supporten.

Enkele voorbeelden

Assertions zijn dusdanig generiek dat ze ook kunnen worden gebruikt om de welbekende primary key (PK) en foreign keys (FK) te declareren. Feitelijk zijn PK's en FK's handige constraint shorthands. Aangezien deze twee type constraints altijd aanwezig zijn, hebben de bedenkers van de SQL taal gemeend om hier speciale taalconstructies voor te introduceren. Een PK zou echter ook (op diverse manieren) als assertion gedeclareerd kunnen worden. Eén hiervan is:

```
create assertion EMP_EMPNO_PK as
check( (select count(distinct EMPNO) from EMP) =
(select count(*) from EMP));
```

NB: we gaan er hierbij van uit dat de EMPNO column al als NOT NULL gedefinieerd is. Een FK zou als assertion als volgt kunnen worden gedeclareerd (we gebruiken nu ook de bekende DEPT table).

```
create assertion EMP_FK_DEPT as
check( not exists(select 'x'
     from EMP e
     where not exists(select 'x'
          from DEPT d
          where d.DEPTNO = e.DEPTNO)));
```

Om de kracht van assertions verder toe te lichten geven we nog twee voorbeelden. Er moeten minstens drie werknemers per afdeling zijn.

```
create assertion NIET_MINDER_DAN_3 as
check( not exists(select e1.DEPTNO
       from (select distinct DEPTNO from EMP) e1
       where 3 > (select count(*)
                  from EMP e2
                  where e2.DEPTNO = e1.DEPTNO)));
```

Afdelingen met een MANAGER moeten ook een CLERK hebben.

```
create assertion ALS_MANAGER_DAN_CLERK as
check( not exists(select 'x'
          from EMP e1
          where e1.job = 'MANAGER'
          and not exists(select 'x'
                  from EMP e2
                  where e2.DEPTNO = e1.DEPTNO
                  and e2.job = 'CLERK')));
```

Ook al worden assertions (nog) niet ondersteund in de DBMS'en van vandaag, dan nog zouden ze als ideale documentatie (of beter nog: specificatie) kunnen dienen om alle constraints in uw database design eenduidig vast te leggen. Met eenduidig bedoel ik, niet ambigu. Documentatie van constraints in gewoon Nederlands loopt namelijk altijd het risico ambigu te zijn: verschillende personen (programmeurs) kunnen de tekst verschillend interpreteren. Assertions bieden een mooi formalisme om:
• die ambiguiteit te vermijden, en
• nog begrijpbaar te zijn voor de (SQL) programmeurs.

Om zelfverzekerd assertions te kunnen specificeren is enige theoretische achtergrond van het relationele model onontbeerlijk. Elementaire verzamelingleer en logica vormen deze (wiskundige) achtergrond. Een goed inzicht in hoe en waar SQL afwijkt van de wiskunde, stelt je in staat om vlot assertions te specificeren. Dit is één van de hoofd onderwerpen van het boek dat ik met Lex de Haan heb geschreven.

Is assertion support complex?

Hoe komt het nou dat SQL assertions nog niet gesupport worden? Ik ben er zeker van dat we allemaal assertions zouden omarmen. U maakt toch ook dankbaar gebruik van de PK's en FK's? Waar het om gaat bij het leveren van support voor assertions, is dat de feature wel praktisch bruikbaar moet zijn. Hiermee bedoel ik dat:

a) de onderliggende implementatie van assertions efficient dient te zijn, en
b) de impact op de vereiste transactie serialisatie minimaal dient te zijn.

Laten we dit even kort bekijken aan de hand van het eerste voorbeeld, de EEN_PRESIDENT assertion (er mag hooguit één president zijn). Een 'luie' implementatie van assertions zou zijn dat het DBMS direct na elke insert, update of delete, de boolean expressie zou evalueren. Dit zou verre van efficiënt zijn, immers als een transactie een CLERK insert, is er geen enkele behoefte om de EEN_PRESIDENT assertion te controleren. Het inserten van een CLERK kan die assertion nooit schenden. Er zijn maar twee situaties (events) die aanleiding geven om de EEN_PRESIDENT assertion te controleren:

1. als een transactie een PRESIDENT toevoegt (een insert), en
2. als een transactie iemand tot PRESIDENT promoveert (een update dus).

En hier zit nu juist de complexiteit. Het heeft weinig zin voor de DBMS leverancier om de luie implementatie te leveren: die zouden wij niet gaan gebruiken vanwege de inefficiëntie. Maar het alternatief, de 'sophisticated' implementatie waarbij het DBMS gegeven een (op voorhand willekeurige) boolean expressie vooraf berekent bij welke events die expressie gecontroleerd moet worden, is een wetenschappelijk complex probleem. Het tweede aspect van 'praktisch bruikbaar' was: serialisatie van transacties. Zonder de nodige serialisatie van transacties zou assertion support onvolledig (en dus incorrect) zijn. U bent wellicht op de hoogte van het feit dat bij FK's het DBMS ook de nodige locks introduceert om bijvoorbeeld het scenario te voorkomen dat transactie A een child-loze parent row verwijdert, terwijl op hetzelfde moment transactie B de eerste child-row introduceert voor die parent. En hierdoor uiteindelijk een child-row zonder bijbehorende parent-row aanwezig is. In het geval van de EEN_PRESIDENT assertion geldt dat er niet twee transacties tegelijk een PRESIDENT mogen introduceren, ofwel door de eerder beschreven insert uit te voeren, ofwel door de eerder beschreven update uit te voeren. Het bepalen van de vereiste serialisatie is onderdeel van het eerder genoemde wetenschappelijk probleem.

Alternatieven

Het zal waarschijnlijk nog wel enige major releases duren alvorens bijvoorbeeld Oracle ons bruikbare support voor assertions levert. In de tussentijd zullen we dus procedureel code moeten implementeren voor alle constraints die niet declaratief afgehandeld kunnen worden. Belangrijk aspect hierbij is om deze code onderhoudbaar te houden. Bij grote database applicaties kan dit namelijk snel uit de hand lopen doordat code om één constraint te implementeren vaak verspreid (en gedupliceerd) wordt over diverse transactionele modules heen. Het beste alternatief is om in je software architectuur één plek te hebben waar constraint code geïmplementeerd wordt. Eind jaren 90 is hiervoor vanuit Oracle Consulting het CDM*RuleFrame framework ontwikkeld. Tegenwoordig (tenzij u nog fervent aanhanger bent van Oracle Designer) biedt het RuleGen framework ook een prima alternatief.

Ir. Toon Koppelaars, RuleGen BV. Hij blogt op TheHelsinkiDeclaration.blogspot.com.