Date's theorie voor muteerbaarheid verrassend eenvoudig

De view mutatie-regel

Frido van Orden

De vorige keer hebben we uitgebreid stilgestaan bij het belang van views voor logische en fysieke gegevensonafhankelijkheid. We zijn echter niet toegekomen aan de enige regel die specifiek over views gaat, namelijk die over het muteerbaar zijn van views. Die komt in dit artikel aan bod.

Een welbekend grapje, niet alleen in de IT-wereld overigens, luidt 'het prettige aan standaarden is dat je er zoveel hebt om uit te kiezen'. Deze conclusie is niet merkwaardig voor wie bedenkt welke economische, organisatorische en commerciële belangen allemaal van invloed zijn bij het tot stand komen van (internationale) standaarden.

Zoveel theorieën, zoveel keus

Hoe anders gaat het toe in de wereld van de wetenschap, waar de ratio de boventoon voert en het onderling vliegen afvangen door wetenschappers in de meeste gevallen leidt tot nog meer en betere wetenschappelijke resultaten, in plaats van bloedgroepenstrijd. Natuurlijk kent de informatica hier weer relatief veel uitzonderingen, denkt u in ons eigen deel-vakgebied maar aan de strijd tussen de relationelen en de object-georiënteerden. Maar ook de o zo degelijke en wetenschappelijk verantwoorde relationelen kunnen er binnen hun eigen wereld wat van: over weinig theoretische database-vraagstukken is zoveel wetenschappelijke literatuur verschenen als over de problematiek van muteerbaarheid van views, en tegelijkertijd lijkt nergens een 'communis opinio' zo ver weg als juist hier. Inmiddels is het onderwerp alweer zo'n jaar of tien 'uit', wat betekent dat we nog erg lang last zullen hebben van dit steentje (of zeg maar gerust flinke kiezel) in de relationele schoen.
Is er dan helemaal geen hoop meer? Jawel, want bij het doorbladeren van wat oude literatuur blijkt dat de relationele peetvader Chris Date al jaren geleden een theorie heeft ontwikkeld voor muteerbaarheid van views, die niet alleen doorwrocht is en leunt op de fundamenten van het relationele model, maar die ook nog eens verrassend eenvoudig is. Wie schrijft die blijft, ruim baan derhalve in dit artikel voor Date's theorie.

Semantiek

We hebben het in deze artikelserie al eerder gehad over het predikaat van een relatie. Dit predikaat definieert wat relaties betekenen. Zo kunnen we bijvoorbeeld de bekende relatie in tabel 1 het volgende predikaat toekennen:
De medewerker met het opgegeven medewerkernummer (Med#) heeft de opgegeven naam (Naam), werkt op de opgegeven afdeling (Afd#) en verdient het opgegeven salaris (Salaris). Als het afdelingsnummer A1 is, is het salaris minder dan 44K. Geen twee medewerkers hebben hetzelfde medewerkernummer. Door in het predikaat waarden in te vullen krijgen we een expressie die 'waar' of 'onwaar' oplevert.
Med#='M1', Naam='Jansen', Afd#='A1', Salaris=25K
levert *waar* op, terwijl
Med#='M1', Naam='Peters', Afd#='A3', Salaris=38K
de waarde *onwaar* oplevert.

Medewerker

Med#	Naam	Afd#	Salaris
M1	Jansen	A1	25K
M2	Bouali	A1	30K
M3	Wilders	A2	80K
M4	Marijnissen	A2	15K

Tabel 1.

Relational Rules (9)

De vorig jaar overleden dr. E.F. Codd werd wereldberoemd met zijn serie publicaties over een gegevens(meta)model dat later bekend zou worden onder de naam 'Relationeel Model'. De eerste uit die serie publicaties was een intern IBM Research Report dat uitkwam in 1969. 2004 zal dus gelden als een lustrum – 35 jaar Relationeel Model.
Frido van Orden schrijft in Database Magazine een serie artikelen over de betekenis en de erfenis van het gedachtengoed van Codd. In deze aflevering wordt regel 6 besproken.
Regel 6, De view mutatie-regel luidt: alle views die theoretisch muteerbaar zijn dienen muteerbaar te zijn door het systeem.

Uiteraard verandert de betekenis van het predikaat als de relatie andere rijen bevat. Toch weten we al een hoop zonder naar de inhoud van de relatie te kijken. Zo weten we bijvoorbeeld dat:
- De waarde voor Med# komt uit het domein van medewerkernummers;
- De waard e voor Naam komt uit het domein van namen;
- De waarde voor Afd# komt uit het domein van afdelingnummers;
- De waarde voor Salaris komt uit het domein van bedragen;
- Als de waarde voor Afd# 'A1' is, dan is het salaris minder dan 44K;
- De waarde voor Emp# is uniek voor alle waarden voor Emp# in de relatie.

Deze kennis wordt door het DBMS gebruikt bij het muteren van de relatie om te valideren dat de gegevens voldoen aan alle integriteitsregels. We kunnen hier een expressie voor definiëren die we het relatiepredikaat zullen noemen:

```
e.Med# IN Med#_dom AND
e.Naam IN Naam_dom AND
e.Afd# IN Afd#_dom AND
e.Salaris IN Bedrag_dom AND
(IF e.Afd# = 'A1' THEN e.Salaris < 44K) AND
(IF e.Med# = f.Med# THEN e.Naam = f.Naam AND
         e.Afd# = f.Afd# AND e.Salaris = f.Salaris)
```

> Natuurlijk mogen views altijd worden gebaseerd op andere views

Het mooie van relatiepredikaten is dat ze niet alleen van toepassing zijn voor basisrelaties (lees voor het gemak: tabellen), maar ook voor afgeleide relaties (lees: views). Nemen we bijvoorbeeld een tabel A met predikaat PA en een tabel B met predikaat PB, dan is het predikaat voor de afgeleide relatie A INTERSECT B natuurlijk PA AND PB, dat wil zeggen dat een rij alleen in de afgeleide relatie voorkomt indien het zowel in A als in B voorkomt. Idem dito luidt voor een simpele restrictie T WHERE *conditie* het predikaat PT AND *conditie*. Soortgelijke afleidingen zijn te maken voor de andere relationele operatoren. We komen daar straks op terug.

Verdere principes

Het uitgaan van relatiepredikaten is verreweg het belangrijkste aspect van Date's theorie. Er zijn er echter nog enkele, waarvan de belangrijkste hier niet onvermeld mogen blijven:
- Of een view muteerbaar is of niet moet een kwestie van semantiek zijn, niet van syntax. Het mag dus bijvoorbeeld niet zo zijn dat de view

```
CREATE VIEW V AS SELECT * FROM Medewerker WHERE
                        Afd# = 'A1' OR Salaris > 33K
```

wel muteerbaar is en de semantisch equivalente view

```
CREATE VIEW V AS (SELECT * FROM Medewerker WHERE
Afd# = 'A1') UNION (SELECT * FROM Medewerker WHERE
                                    Salaris > 33K)
```

niet. De huidige SQL-standaard denkt hier bijvoorbeeld heel anders over: de eerste view is muteerbaar, de tweede niet;
- Views die overeenkomen met een tabel moeten natuurlijk muteerbaar zijn, en derhalve ook semantisch equivalente views als A UNION A, A INTERSECT A, A MINUS B (indien B geen rijen uit A bevat) en A WHERE true;
- Het gedrag van de mutatie op een view moet waar mogelijk symmetrisch zijn. Indien we bijvoorbeeld een rij verwijderen uit de view V =A INTERSECT B, dan mag het niet zo zijn dat arbitrair rijen uit A maar niet uit B worden verwijderd, hoewel dat op zich wel een correct resultaat zou opleveren;
- Het resultaat van een UPDATE moet gelijk zijn aan het resultaat van een DELETE gevolgd door een INSERT;
- INSERTS op de view worden afgebeeld op INSERTS in onderliggende tabellen en DELETE's op DELETE's (voor UPDATE's: zie het vorige punt). De reden hierachter is dat vrijwel alle relationele operatoren dusdanig kunnen worden gebruikt dat ze effectief niets doen (bijvoorbeeld A UNION A, of A WHERE true). Indien voor een bepaald soort views, bijvoorbeeld union views, zou gelden dat bijvoorbeeld INSERT's op DELETE's worden afgebeeld, dan zou dit ook moeten gelden voor A UNION A, en daarmee (want semantisch equivalent) ook voor INSERTs op A zelf, ofwel rechtstreeks op de tabel (!);
- Natuurlijk mogen views altijd worden gebaseerd op andere views. Waar kortweg gesproken wordt over het verwerken van een mutatie op een view in de onderliggende tabel, geldt dit natuurlijk ook voor het verwerken in onderliggende views.

Met de bovenstaande principes in het achterhoofd is het niet moeilijk meer om een te definieren hoe een view moet worden gemuteerd voor alle elementaire relationele operatoren Union, Intersection, Difference, Restriction, Projection, Extension en Join. Daarmee kunnen we tevens alle views aan die gebaseerd zijn op een combinatie van deze operatoren.
Zoals eerder gezegd hoeven we alleen INSERT's en DELETE's te definiëren, de betekenis van UPDATE kan hieruit worden afgeleid. Daarnaast geldt natuurlijk dat de rijen die worden toegevoegd of verwijderd nooit tot schending van het relatiepredikaat van de view mogen leiden. Toevoegen van een rij met waarde x=70 aan de view (T WHERE x > 50 INTERSECT T WHERE x < 60) is dus altijd verboden! (anders dan in SQL, waar expliciet WITH CHECK OPTION bij creëren van een view moet worden gedefinieerd om schending van het relatiepredikaat te verbieden).

Unions, Intersections en Differences

We beginnen met de Union. Een rij komt voor in de view A

UNION B indien de rij ofwel in A, ofwel in B, ofwel in allebei voorkomt. Voegen we een rij toe in A UNION B, dan moeten we dus diezelfde rij toevoegen in A, of in B, of in allebei. We kunnen de rij toevoegen in A indien de rij voldoet aan het relatiepredikaat van A (notatie verder: PA).
Idem dito geldt natuurlijk voor B. Echter: het toevoegen van de rij in A kan als *side effect* hebben dat de rij ook in B is toegevoegd (bijvoorbeeld in geval van T UNION T, of (T WHERE x < 100) UNION (T WHERE x > 50)). In dat geval zou een rij die zowel aan PA als PB voldoet 2 keer worden toegevoegd, met mogelijk zelfs een schending van een unieke sleutel als gevolg.
Daarom wordt de rij alleen aan B toegevoegd indien dit niet al automatisch als side effect van het toevoegen aan A is gebeurd (en natuurlijk alleen indien de rij aan PB voldoet).
Bij verwijderingen geldt iets soortgelijks. Indien een rij verwijderd wordt uit A UNION B, en de rij komt voor in A, dan wordt de rij verwijderd uit A (met als mogelijk side effect verwijdering uit B). Komt de rij (nog steeds) voor in B, dan wordt de rij verwijderd uit B.

Als we dit begrijpen zijn Intersections niet moeilijk meer. Bij toevoegen van een rij in A INTERSECT B wordt de rij aan A wordt toegevoegd indien de rij nog niet in A voorkwam (met mogelijk side effect van toevoegen in B). Indien de rij (nog steeds) niet in B voorkomt wordt de rij aan B toegevoegd. Bij verwijderen uit A INTERSECT B wordt de rij uit A verwijderd (met mogelijk side effect van verwijderen uit B). Indien de rij (nog steeds) voorkomt in B wordt deze uit B verwijderd.
Merk overigens op dat INSERT en DELETE hier niet exact elkaars tegengestelde zijn. Als A en B verschillende tabellen zijn dan kan het toevoegen van een rij in A INTERSECT B leiden tot toevoegen in A, maar niet in B omdat de rij daarin al aanwezig was. Vervolgens verwijderen van dezelfde rij uit A INTERSECT B leidt vervolgens tot verwijdering uit zowel A als B. De omgekeerde volgorde (eerst verwijderen, dan toevoegen) is overigens wel symmetrisch. Date's pragmatische argument tegen deze schending van zijn eigen principes is hier dat dergelijke situaties alleen voorkomen in slecht ontworpen databases waar identieke rijen voorkomen in twee verschillende (basis)tabellen.
Voor de difference view A MINUS B geldt dat de rijen in A voorkomen maar niet in B. Bij toevoegen van een rij aan A MINUS B wordt de rij dus toegevoegd aan A maar niet aan B. Bij verwijdering van de rij wordt de rij verwijderd uit A.

Restricties, Projecties en Extensies

Restricties zijn uiterst eenvoudig. Een rij die wordt toegevoegd in A WHERE *conditie* moet uiteraard voldoen aan de conditie en wordt toegevoegd in A. Bij verwijdering wordt de rij verwijderd uit A.
Projecties lijken ook eenvoudig maar zijn toch iets ingewikkelder. Wat moet er bijvoorbeeld gebeuren als we een rij toevoegen in de volgende view?

```
CREATE VIEW V AS SELECT Med#, Naam FROM Medewerker
```

Het is duidelijk dat er een rij moet worden toegevoegd in Medewerker, en de waarden voor Med# en Naam zijn ook duidelijk. Maar welke waarde moet er in Afd# en Salaris worden ingevuld? De enige waarde die in aanmerking komt is de standaard (default) waarde die voor deze kolommen is gedefinieerd. Voor niet verplichte kolommen is er de impliciete standaardwaarde NULL, en voor iedere kolom kan een expliciete standaardwaarde worden gespecificeerd. Indien bijvoorbeeld Afd# de standaardwaarde 'A1' heeft en Salaris de standaardwaarde 0, dan leidt het toevoegen van de rij Med#=M5, Naam=Polderman aan de view V, tot het toevoegen van de rij Med#=M5, Naam=Polderman, Afd#=A1, Salaris=0 aan Medewerker. Indien een kolom verplicht is, geen standaardwaarde heeft en niet is opgenomen in de view, dan is het niet mogelijk om rijen toe te voegen via de view.

> Natuurlijk kent de informatica weer relatief veel uitzonderingen

Bij verwijderen van rijen uit de view worden alle rijen uit de onderliggende tabel verwijderd met overeenkomstige waarden. Let op: indien de view niet alle sleutelkolommen bevat betekent dit dat het verwijderen van 1 rij uit de view kan leiden tot verwijdering van meerdere rijen uit de onderliggende tabel.
Definiëren we bijvoorbeeld de view

```
CREATE VIEW V AS SELECT DISTINCT Afd# FROM
                                        Medewerker
```

en verwijderen we vervolgens een rij met

```
DELETE FROM V WHERE Afd#='A1'
```

dan worden alle rijen met Afd#='A1' uit Medewerker verwijderd. Merk op dat de view is aangemaakt met het SQL DISTINCT sleutelwoord. Volgens de relationele theorie bestaat er niet zoiets als de constructie die in SQL 'SELECT ALL' of kortweg 'SELECT' (zonder DISTINCT dus) heet: de selectie SELECT Afd# FROM Medewerker levert in ons voorbeeld dus twee rijen op (met waarden A1 respectievelijk A2) en niet twee rijen met waarde A1 en twee rijen met waarde A2
Extensies komen u wellicht niet bekend voor, maar het is niets meer dan de relationele term voor het uitbreiden van een selectieresultaat met een willekeurige expressie. Het afbeelden van views met extensies op de onderliggende tabellen is natuurlijk triviaal.

Joins

Het updaten van joins is altijd een heet hangijzer geweest. Een belangrijke reden hiervoor is dat het bij joins vaak niet mogelijk is om een rij te verwijderen uit de join zonder dat automatisch ook andere rijen worden verwijderd. Omdat dit anders is bij verwijderingen uit tabellen, waar dergelijke side effects niet optreden, is er altijd voor gekozen om dergelijke views niet muteerbaar te verklaren.

De benadering van Date is anders. Date wijst side effects niet af. Immers, het muteren rechtstreeks op de basistabellen leidt ook tot side effects, namelijk op de op de basistabel gebaseerde views. Het blijkt dat als we side effects accepteren dat alle joins muteerbaar zijn! Dit illustreren we met een paar voorbeelden op de bekende Supplier-Parts database, zie tabel 2, 3 en 4.

Supplier

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Tabel 2.

Part

P#	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Tabel 3.

SupplierPart

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Tabel 4.

De algemene regels bij het updaten van joins A JOIN B luiden aldus:

- Bij toevoegen wordt 'het A-deel' van de rij toegevoegd aan A indien het daar nog niet voorkwam. Indien 'het B-deel' van de rij niet voorkomt in B wordt dit in B toegevoegd;
- Bij verwijderen wordt 'het A-deel' van de rij verwijderd uit A en 'het B-deel' verwijderd uit B

Supplier JOIN SupplierPart (SSP)

S#	SNAME	STATUS	CITY	P#	QTY
S1	Smith	20	London	P1	300
S1	Smith	20	London	P2	200
S1	Smith	20	London	P3	400
S1	Smith	20	London	P4	200
S1	Smith	20	London	P5	100
S1	Smith	20	London	P6	100
S2	Jones	10	Paris	P1	300
S2	Jones	10	Paris	P2	400
S3	Blake	30	Paris	P2	200
S4	Clark	20	London	P2	200
S4	Clark	20	London	P4	300
S4	Clark	20	London	P5	400

Tabel 5.

Nemen we als voorbeeld een 1:n join tussen Supplier en SupplierPart (over de kolommen S#), zie tabel 5:

- Indien we de rij <S4, Clark, 20, London, P6, 100> aan de view SSP toevoegen, dan leidt dit tot het toevoegen van de rij <S4, P6, 100> aan SupplierPart;
- Indien we de rij <S6, Green, 20, London, P6, 100> aan de view SSP toevoegen, dan leidt dit tot het toevoegen van de rij <S6, Green, 20, London> aan Supplier en toevoegen van de rij <S6, P6, 100> aan SupplierPart;
- Toevoegen van de rij <S4, Clark, 20, Athens, P6, 100> geeft een foutmelding: er kan geen rij <S4, Clark, 20, Athens> aan Supplier worden toegevoegd omdat er al een andere rij in Supplier bestaat met dezelfde waarde voor S#;
- Verwijderen van de rij <S3, Blake, 30, Paris, P2, 200> uit SSP leidt tot verwijdering van de rij <S3, Blake, 30, Paris> uit Supplier en verwijdering van de rij <S3, P2, 200> uit SupplierPart;
- Verwijderen van de rij <S1, Smith, 20, London, P1, 300> uit SSP leidt tot verwijdering van de rij <S1, Smith, 20, London> uit Supplier en verwijdering van de rij <S1, P1, 300> uit SupplierPart. Merk op dat er nog steeds rijen in SupplierPart voorkomen met S#=S1! Deze zullen alsnog moeten worden verwijderd om de transactie te doen slagen.

Voor een m:n join gelden soortgelijke regels. Nemen we als voorbeeld de join tussen Supplier en Part over de kolom City, zie tabel 6:

- Indien we de rij <S7, Brown, 15, Oslo, P8, Wheel, White, 25> aan de view SCP toevoegen, dan leidt dit tot het toevoegen van de rij <S7, Brown, 15, Oslo> aan Supplier en toevoegen van de rij <P8, Wheel, White, 25, Oslo> aan Part;
- Indien we de rij <S1, Smith, 20, London, P7, Washer, Red 5> aan de view SCP toevoegen, dan leidt dit tot het toevoegen van de rij <P7, Washer, Red, 5, London> aan Part. Hiermee wordt

Supplier JOIN Part (SCP)

S#	SNAME	STATUS	CITY	P#	PNAME	COLOR	WEIGHT
S1	Smith	20	London	P1	Nut	Red	12
S1	Smith	20	London	P4	Screw	Red	14
S1	Smith	20	London	P6	Cog	Red	19
S2	Jones	10	Paris	P2	Bolt	Green	17
S2	Jones	10	Paris	P5	Cam	Blue	12
S3	Blake	30	Paris	P2	Bolt	Green	17
S3	Blake	30	Paris	P5	Cam	Blue	12
S4	Clark	20	London	P1	Nut	Red	12
S4	Clark	20	London	P4	Screw	Red	14
S4	Clark	20	London	P6	Cog	Red	19

Tabel 6.

als side effect ook de rij <S4, Clark, 20, London, P7, Washer, Red, 5> aan de view toegevoegd;

- Verwijderen van de rij <S1, Smith, 20, London, P1, Nut, Red, 12> leidt tot verwijderen van de rij <S1, Smith, 20, London> uit Supplier en verwijderen van de rij <P1, Nut, Red, 12, London> uit Part, en daarmee tot verwijdering van in totaal vier rows uit de view, namelijk alle rows met ofwel S#=S1, ofwel P#=P1.

Aggregaties

We zijn nog 1 belangrijke categorie relationele operatoren vergeten, namelijk de aggregaties (sommaties, gemiddelden, minimum/maximum etcetera). Ook hier is het 'common sense' dat views met aggregaties niet muteerbaar zijn. Toch is ook dat maar ten dele waar. Nemen we bijvoorbeeld de volgende view.

```
CREATE VIEW V AS
SELECT P#, SUM(QTY) TOTAL_QTY
FROM SupplierPart
GROUP BY P#
```

Het is duidelijk dat toevoegen van een rij aan de view voor problemen zorgt, omdat we geen waarde voor S# hebben. Indien op SupplierPart geen default-waarde voor S# is gedefinieerd (en dat is in dit geval zeer waarschijnlijk) dan kunnen we geen rijen toevoegen aan de view. Verwijderen kan echter wel, zo leidt verwijderen van de rij <P1, 600> uit de view tot het verwijderen van de twee rijen uit SupplierPart met P#=P1.

Conclusie

Het blijkt dat er op het gebied van muteerbaarheid van views veel meer mogelijk is dan vaak wordt aangenomen. Bovendien blijkt de semantiek verrassend eenvoudig te zijn, bepaald niet onbelangrijk gezien het in de praktijk nog steeds hoge 'zwarte magie' gehalte van geavanceerde relationele en SQL-expressies.

Literatuur

Chris Date, 'Updating Union, Intersection, and Difference Views', en 'Updating Joins and Other Views', in: Relational Database Writings 1991-1994, Addison Wesley.

Frido van Orden (frido.van.orden@faapartners.com) is partner bij FAA Partners.