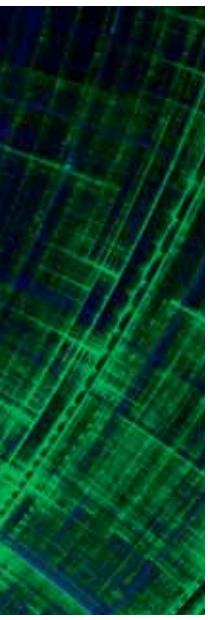


Gegevens met onvolkomenheden zijn eenvoudig op te slaan

Onzekere databases

Maurice van Keulen

Een recente ontwikkeling in het databaseonderzoek betreft de zogenaamde 'onzekere databases'. Dit artikel beschrijft wat onzekere databases zijn, hoe ze gebruikt kunnen worden en welke toepassingen met name voordeel zouden kunnen hebben van deze technologie.

Onzekere databases zijn database management systemen die grote hoeveelheden onzekere gegevens kunnen opslaan, queryen en updaten. De technologie begint schoorvoetend het laboratoriumstadium te ontgroeien. Inmiddels zijn er enkele goed werkende prototypes beschikbaar. In de IT pretenderen we doorgaans dat informatiesystemen perfecte weerspiegelingen zijn van een perfecte wereld. Er wordt namelijk gewerkt onder de aanname dat gegevens opgeslagen in databases volledig en correct overeenkomen met de werkelijkheid.

Deze idealistische aanname is natuurlijk maar zelden waar, want gegevens bevatten doorgaans de volgende onvolkomenheden:

- Fouten. Bijvoorbeeld typefouten of conversiefouten;
- Missende gegevens;
- Onnauwkeurige of bedrieglijk exacte gegevens. Een meetinstrument of sensor produceert metingen met een bepaalde precisie. Die metingen worden vaak opgeslagen zonder informatie over de omstandigheden en de behaalde precisie;
- Inconsistente gegevens. Verschillende gegevens kunnen elkaar tegenspreken of bepaalde integriteitsregels of bedrijfsregels overtreden.

Discreet en continu

Het is natuurlijk niet zo dat dergelijke problemen volledig worden genegeerd. Men voert veelal handmatige en automatische controles uit om dergelijke onvolkomenheden te detecteren en te repareren. Maar dit is doorgaans een apart geïsoleerd proces; in het gebruik van de informatie hanteren we de idealistische aanname dat alles wel volledig is en klopt. In een wereld waar informatiesystemen meer en meer worden gekoppeld, we steeds meer afhankelijk worden van gegevens van goede kwaliteit en de hoeveelheid informatie alleen maar harder groeit, loopt het met deze aanpak een keer spaak.

Imperfectie is niet noodzakelijkerwijs iets slechts; eerder een 'fact of life'. De uitdaging is om informatiesystemen soepeler met onvolkomenheden om te kunnen laten gaan. Dat is niet alleen een kwestie van controles en data cleaning, maar ook van robuustheid en flexibiliteit. Alle bovengenoemde soorten onvolkomenheden zijn te zien als *onzekerheid*: we weten simpelweg op een bepaald moment niet hoe het werkelijk zit. Door eerlijk en expliciet met die onzekerheid om te gaan, kan een informatiesysteem zich daaraan aanpassen en goed genoeg functioneren ondanks dat niet alle gegevens precies en correct beschikbaar zijn.

Onzekere databases maken het mogelijk om eerlijk en expliciet met onzekerheid om te gaan, zonder dat het de complexiteit van de informatiesystemen significant verhoogt. Kort gezegd, een onzekere database maakt het mogelijk om niet alleen de data maar ook de onzekerheid over die data expliciet op te slaan. Vervolgens zijn die gegevens op een normale manier te queryen. Je krijgt alleen soms *onzekere antwoorden* als de onderliggende gegevens onzeker zijn. Onder de motorkap zitten weliswaar ingewikkelde opslagstructuren en algoritmes om efficiënt die onzekere data te kunnen verwerken, maar aan de buitenkant is het simpelweg SQL met een paar uitbreidingen.

Er is een onderscheid te maken tussen discrete en continue onzekerheid. We praten over discrete onzekerheid als er een beperkt aantal gevallen onderscheiden kan worden, bijvoorbeeld klant A woont ofwel op adres X ofwel op adres Y. We praten over continue onzekerheid als een bepaalde numerieke waarde niet precies is, bijvoorbeeld een temperatuursensor heeft 27,4 graden Celsius gemeten, maar die is maar tot plusminus 0,5 graad Celsius nauwkeurig. Voorbeelden van open source prototypes die discrete onzekerheid ondersteunen zijn MayBMS[1] en Trio[2]. Voorbeelden van prototypes die ook continue onzekerheid ondersteunen zijn Orion[3] en MystiQ[4]. We richten ons in dit artikel echter voornamelijk op discrete onzekerheid.

Applicatiegebieden waar onzekere databases met name goed toegepast kunnen worden, zijn natuurlijk die waar de data inherent onzeker zijn, dat wil zeggen veel ambiguïteiten, hiaten en onnauwkeurigheden bevatten. Denk bijvoorbeeld aan sensor data management, automatische informatie extractie uit ongestructureerde tekst, data-integratie en het automatisch taggen van pagina's, berichten en multimedia. Ook applicatiegebieden waar voorspellingen een rol spelen zijn geschikt, zoals decision support, risk management, surveillance en forensics, en fraude en plagiaatdetectie.

Werking onzekere database

Eerst leggen we uit hoe onzekere gegevens worden opgeslagen. Daarna wordt een stukje theorie inzichtelijk gemaakt met de analogie van de 'parallel universes' uit de science-fiction. Later gaan we in op hoe de onzekere gegevens zijn te queryen met SQL.

Afbeelding 1 geeft een voorbeeld van een tabel met onzekerheid. De tabel bevat informatie over de klanten van een bepaald bedrijf, toevallig betreft het gegevens over één familie. Ten eerste is de naam in het record 7 onzeker: het is ofwel 'Opa Jansen' ofwel 'Opa Janssen', maar men weet niet helemaal zeker welke de juiste is. Deze onzekerheid over een attribuutwaarde kunnen we opslaan als twee *alternatieven*. Waarschijnlijk is er ooit geconstateerd dat er twee varianten voor de naam van die klant de ronde deden, maar dat het zeker om één persoon ging.

Aangezien de rest van de klanten allemaal 'Jansen' heet is vermoedelijk die naam de juiste. Dit is uitgedrukt door aan de waarde 'Opa Jansen' een waarschijnlijkheid van 90 procent toe te kennen en aan de andere waarde de resterende 10 procent.

Bij 'Oom Jansen' (record 5) is het adres en de leeftijd onzeker: met 80 procent zekerheid is hij 31 en woont op 'Laan 5', maar hoewel onwaarschijnlijk (waarschijnlijkheid 20 procent) zou hij ook 29 kunnen zijn en op 'Straat 3' wonen. Dit is een geval waarbij meerdere attribuutwaarden onzeker zijn. Het zou immers zo kunnen zijn dat 'Oom Jansen' twee keer iets gekocht heeft met verschillende adresgegevens, waarbij het wel zeker is dat het om één en dezelfde persoon ging, maar niet welk adres en welke leeftijd correct is. Wel vermoeden we dat 'Laan 5' correct is, want dat lijken recentere gegevens omdat de leeftijd hoger is. Dit vermoeden is uitgedrukt in de hogere waarschijnlijkheid (80 procent) voor 'Laan 5'.

Op 'Laan 5' woont ook ene 'Baby Jansen' van 1 jaar, maar het is niet zeker of hij/zij wel een klant is (waarschijnlijkheid 70 procent). Waar in het vorige voorbeeld de waarschijnlijkheden optellen tot 100 procent, geldt dat dus voor dit record niet.

Deze vorm van onzekerheid wordt ook wel een *maybe-record* genoemd: het bestaan van het hele record is onzeker.

Afhankelijkheden

Een belangrijk detail bij het omgaan met onzekere gegevens en waarschijnlijkheden zijn de *afhankelijkheden*. De onzekerheden die we gezien hebben in records 5, 6 en 7 zijn onafhankelijk van elkaar. De wiskunde achter de kansrekening is heel expliciet en exact over wat een afhankelijkheid betekent en wat de consequenties daarvan zijn. In ons geval komt het er grofweg op neer dat het feit of het in werkelijkheid 'Opa Jansen' of 'Opa Janssen' is, geen invloed heeft op, bijvoorbeeld, of 'Baby Jansen' nu wel of niet een klant is. Alle vier combinaties zijn mogelijk. Voor de onzekerheid binnen record 5 geldt dat niet: niet alle combinaties van adres, postcode en leeftijd zijn mogelijk voor 'Oom Jansen'. We zeggen dat de alternatieven voor de drie attribuutwaarden van elkaar afhankelijk zijn: ofwel voor alle drie geldt de bovenste waarde, ofwel voor alle drie de onderste.

Tenslotte moet nog vermeld worden dat de eerder genoemde prototypes MayBMS en Trio de onzekerheden in de tabellen op andere wijze laten zien. Het komt echter op hetzelfde neer; voor de duidelijkheid hebben we voor deze presentatie gekozen.

Onzekere databases zijn gebaseerd op een elegante en simpele theorie die het niettemin mogelijk maakt om hele krachtige algoritmen te ontwikkelen voor het efficiënt queryen van grote hoeveelheden onzekere data. Eerder zagen we al dat alternatieven van verschillende records onafhankelijk waren en dat dus alle combinaties mogelijk zijn. Dat kun je ook doen voor de hele database. In het geval van afbeelding 1 zijn er 2 x 2 x 2 = 8 com-

id	Naam	adres	postcode	leeftijd	
1	Jan Jansen	Straat 1	7599 AA	50	1.0
2	Marieke Jansen	Straat 1	7599 AA	48	1.0
3	Kleuter Jansen	Straat 1	7599 AA	6	1.0
4	Tiener Jansen	Straat 1	7599 AA	12	1.0
5	Oom Jansen	Straat 3	7599 AA	29	0.2
		Laan 5	7588 BB	31	0.8
6	Baby Jansen	Laan 5	7588 BB	1	0.7 ?
7	Opa Jansen	Laan 8	7588 BB	70	0.9
	Opa Janssen				0.1

Afbeelding 1: Voorbeeldtabel Klant.

binaties (twee mogelijkheden voor elk van de records 5, 6 en 7; voor de overige records is er maar één mogelijkheid). Er zijn dus acht mogelijke toestanden waarin de database zich zou kunnen bevinden, elk met een bepaalde waarschijnlijkheid. Dit noemen we de *mogelijke werelden* (possible worlds). De tabel in afbeelding 1 is te zien als een compacte manier van opslaan van alle mogelijke databases.

Op deze manier bekeken is het meteen duidelijk wat *het* antwoord op een query moet zijn. Je zou namelijk in gedachten de query kunnen evalueren op alle individuele mogelijke databases. Op die manier krijg je alle mogelijke antwoorden. Getoond op een soortgelijke compacte manier is dat dus *het* onzekere antwoord op de query (zie afbeelding 2).
Dit is theorie; een implementatie zou het natuurlijk nooit op deze manier doen, want het aantal mogelijke werelden groeit exponentieel. De theorie geeft echter een waarschijnlijkheidsinterpretatie aan een onzekere database die het mogelijk maakt om alle technieken uit de kansrekening en statistiek toe te passen om efficiënt query's rechtstreeks op de compacte representatie uit te voeren (de dubbele pijl in afbeelding 2) waarbij er gegarandeerd het juiste antwoord uit komt.

Query's

Het mooie van een onzekere database is dat je deze gewoon met SQL kunt queryen zonder veel kennis van kansrekening en statistiek. We geven enkele voorbeelden. Merk op dat voorbeeld 1 en 2 precies zo zijn geformuleerd zoals je dat voor een normale database ook zou doen. Het enige verschil zit er in dat het resultaat van de query onzeker kan zijn.
De genoemde systemen hebben SQL uitgebreid met extra faciliteiten om met onzekere data om te gaan. Helaas is er op dit vlak nog geen standaardisatie. De vier voorbeelden gebruiken de SQL-variant van MayBMS. Voorbeeld 3 gebruikt bijvoorbeeld 'ecount' om de verwachtingswaarde uit te rekenen. Voorbeeld 4 gebruikt 'repair key' om onzekere data te construeren. Dat laatste gaat in Trio met behulp van een aangepast 'INSERT'-statement waarmee de alternatieven en waarschijnlijkheden direct kunnen worden opgegeven. Dit voorbeeld illustreert tevens dat het toevoegen, wijzigen en updaten ook op nagenoeg dezelfde manier gaat.

Voorbeeld 1: Simpel SELECT-FROM-WHERE. De namen van alle klanten met postcode 7588 BB.

```
SELECT naam FROM klant WHERE postcode='7588 BB'
```

naam	
Oom Jansen	0.8 ?
Baby Jansen	0.7 ?
Opa Jansen	0.9
Opa Janssen	0.1

Voorbeeld 2: Aggregaten. Per postcode hoeveel klanten die postcode hebben.

```
SELECT postcode, COUNT(*)
FROM klant
GROUP BY postcode
```

postcode	COUNT(*)	
7599 AA	4	0.8
	5	0.2
7588 BB	1	0.06
	2	0.38
	3	0.56

Voorbeeld 3: Verwachtingswaarde. Per postcode, de verwachtingswaarde van het aantal klanten met die postcode.

```
SELECT postcode, ECOUNT(*)
FROM klant
GROUP BY postcode
```

postcode	ECOUNT(*)	
7599 AA	4.2	1.0
7588 BB	2.5	1.0

Voorbeeld 4: Onzekere informatie creëren. We maken een tabel 'inkomen' aan die resultaten van een marktonderzoek bevat, namelijk per postcode en inkomenscategorie, de hoeveelheid respondenten die aangegeven hebben dat ze in die inkomenscategorieën vallen.

```
CREATE TABLE inkomen (
postcode VARCHAR(6),
cat VARCHAR(20), /*laag,modaal,bovenmodaal,hoog*/
respondenten INT
);

INSERT INTO inkomen VALUES ('7599AA','laag',5000);
INSERT INTO inkomen VALUES ('7599AA','modaal',7500);
INSERT INTO inkomen VALUES ('7599AA',
                            'boven modaal',2000);
INSERT INTO inkomen VALUES ('7599AA','hoog',500);
INSERT INTO inkomen VALUES ('7588BB','laag',1000);
INSERT INTO inkomen VALUES ('7588BB','modaal',2000);
INSERT INTO inkomen VALUES ('7588BB',
                            'boven modaal',3000);
INSERT INTO inkomen VALUES ('7588BB','hoog',7000);
```

Het is echter handiger om deze gegevens te zien als onzekerheid over wat de inkomenscategorie is van een bepaalde postcode. Om in termen van de kansrekening te spreken, de data vormen

eigenlijk een kansverdeling. We kunnen dit in MayBMS als volgt aanpakken:

```
CREATE TABLE uinkomen AS
REPAIR KEY postcode IN inkomen WEIGHT BY
                                        respondenten;
```

Het 'repair key' statement maakt het attribuut 'postcode' tot een key van de tabel 'inkomen'. Als er meerdere records zijn met dezelfde waarde voor dat attribuut, dan worden die tot één record samengevoegd; daar waar attribuutwaarden niet overeen komen, ontstaan alternatieven.

Vervolgens koppelen we de informatie als volgt aan onze klanttabel.

```
CREATE TABLE uklant AS
SELECT naam, adres, C.postcode, leeftijd, cat
FROM klant C, uinkomen I
WHERE C.postcode=I.postcode;
```

Wat is nu, bijvoorbeeld, de meest waarschijnlijke inkomenscategorie voor 'Opa Jansen'?

```
select argmax(cat,w)
from (select cat, conf() as w
from uklant
where naam='Opa Jansen'
group by cat) as T;
```

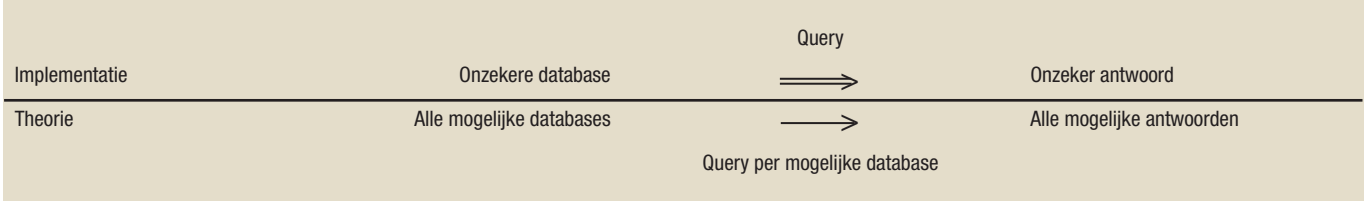
De subquery 'T' bepaalt per inkomenscategorie de waarschijnlijkheid 'w' dat 'Opa Jansen' daarin valt. Hierin wordt impliciet meegenomen dat de klant mogelijk 'Opa Janssen' heet en niet 'Opa Jansen'. Met 'argmax' bepalen we vervolgens de categorie met maximum 'w'. Het resultaat is 'hoog'.

Een blik onder de motorkap

Zowel MayBMS als Trio zijn gebouwd bovenop PostgreSQL. De data worden opgeslagen in normale tabellen, aangevuld met speciale attributen en tabellen die de waarschijnlijkheden, alternatieven en afhankelijkheden registreren. Beide systemen doen dit op een zeer verschillende manier.
Hoewel beide systemen een enigszins verschillende SQL taal aanbieden, zijn ze beide gebaseerd op het vertalen van die query's naar nieuwe SQL query's die worden uitgevoerd op de onderliggende tabellen. Beide volgen de theorie van mogelijke werelden, dus hoewel ze intern verschillend werken, het antwoord op eenzelfde query is hetzelfde.
De vertaalde query's zorgen ook voor de berekening van de waarschijnlijkheden van de alternatieven in het antwoord. Met name dit aspect is lastig én correct én efficiënt te krijgen voor sommige SQL operaties zoals DISTINCT en aggregaten. Een actieve groep databasewetenschappers is bezig met het bedenken van nieuwe algoritmen hiervoor. Een veelbelovende aanpak is om de waarschijnlijkheden te gaan schatten in die gevallen waarvoor het exact berekenen te duur is vanwege de grote hoeveelheden alternatieven. Die schattingen komen dan vaak met nauwkeurigheidsgaranties, bijvoorbeeld de waarschijnlijkheid voor een bepaald alternatief is 37,43 procent met een afwijking van maximaal 0,01 procent. Voor de meeste applicaties is dit voldoende.

MayBMS is intern gebaseerd op 'random variables', een concept uit de statistiek en meestal aangeduid met de letter X. Random variables zijn onderling onafhankelijk en de verschillende waarden die aan een random variable toegekend kunnen worden sluiten elkaar uit. In MayBMS wordt voor elk geval van onzekerheid een nieuwe random variable aangemaakt waarbij elk alternatief een verschillende toekenning krijgt (bijvoorbeeld 'X=1' voor het eerste alternatief, enzovoort). De data zelf worden geannoteerd met de relevante toekenningen (de zogenaamde 'world set descriptor'). In afbeelding 1 zijn er drie gevallen van onzekerheid en dus spelen drie random variables een rol, zeg X, Y en Z. De beide alternatieven van record 5 zijn geannoteerd met X=1 en X=2. Hierdoor sluiten deze alternatieven elkaar uit, omdat X niet tegelijkertijd gelijk kan zijn aan 1 en aan 2. De waarschijnlijkheden zijn aan de toekenningen gekoppeld: de kans op 'X=1' is gelijk aan 0.8.
Doordat een record met meerdere random variables geannoteerd kan zijn, kunnen ingewikkelde afhankelijkheden worden gerepresenteerd. Dit is belangrijk om op alle query's het correcte antwoord te kunnen geven, dat wil zeggen zoals voorgeschreven door de mogelijke werelden theorie. Neem bijvoorbeeld de onderstaande query (wie heeft dezelfde postcode als 'Opa Jansen'?).

```
select K.naam
from klant Opa, klant K
where Opa.naam = 'Opa Jansen'
and K.postcode = Opa.postcode;
```



Afbeelding 2: Relatie tussen de implementatie en de theorie van het queryen van een onzekere database.

Een databasekenner herkent hier gelijk een 'self-join' in, dat wil zeggen een join van de tabel 'klant' met zichzelf. Als je deze query op een naïeve manier zou evalueren, zonder precies zorg te dragen voor de afhankelijkheden, dan krijg je ook 'Opa Janssen' in het antwoord. Dat dit incorrect is, kunnen we zien door alle mogelijke werelden in gedachten af te lopen: in de werelden waar Opa 'Jansen' heet, bestaat er geen 'Opa Janssen' dus die kan niet voorkomen in het antwoord; in de werelden waar Opa 'Janssen' heet is het resultaat leeg.

In Trio worden de afhankelijkheden anders opgeslagen, namelijk met behulp van *lineage*, dat wil zeggen een registratie van op welke andere alternatieven een alternatief gebaseerd is (en dus van afhankelijk is). Dit resulteert in andere algoritmen voor het uitsluiten van incorrecte antwoorden en het berekenen van waarschijnlijkheden. Naar mijn mening is de aanpak van MayBMS generieker en krachtiger. Een benchmark voor onzekere databases is er nog niet, dus het is nog onduidelijk hoe de verschillende systemen zich verhouden qua performance en schaalbaarheid.

Waarom een onzekere database gebruiken?

Applicatiegebieden waar onzekere databases met name goed toegepast kunnen worden, zijn die waar de data inherent onzeker zijn of waar voorspellingen een rol spelen. In de inleiding noemden we al een heel aantal applicatiegebieden. Deze applicatiegebieden bestaan natuurlijk al langer dan vandaag en in elk gaat men ook al in meer of mindere mate effectief om met de onzekerheid omtrent de data. Veelal spelen speciale statistische pakketten of programmabibliotheken daarbij een rol. Wat voegt een onzekere database daar dan nog aan toe?

Er zijn veel redenen om voor databasetechnologie te kiezen. Een onzekere database neemt de verantwoordelijkheid voor het efficiënt en veilig beheren van de onzekerheid omtrent data over. Daar waar omwille van imperfecte informatie geen database wordt gebruikt, kan dat nu wel. En in gevallen waar al voor databasetechnologie gekozen is, kan men effectiever en eenvoudiger met imperfecte informatie omgaan.

Gegevenskwaliteit

In het begin van dit artikel schreef ik dat onvolkomenheden in data te zien zijn als onzekerheid. Men kan daarom geneigd zijn te denken dat we nu gegevenskwaliteit kunnen meten door simpelweg de hoeveelheid onzekerheid te meten. Daarvoor bestaan namelijk meerdere maten, zoals bijvoorbeeld de *entropie*. De kwestie ligt echter subtieler. Voor mij is de kwaliteit van data hoger als die 'dichter bij de waarheid' ligt. Mijn intuïtie is dat een antwoord beter is naarmate de waarschijnlijkheid hoger is waarmee het systeem het *werkelijk juiste antwoord* durft te geven. Neem het beschreven query voorbeeld 1 en stel dat het werkelijk juiste antwoord 'Oom Jansen' en 'Opa Jansen' is. Het foute antwoord 'Baby Jansen' is eigenlijk maar voor 70 procent fout.

Door geen absolute antwoorden te hoeven geven, maar expliciet alternatieven en waarschijnlijkheden mee te nemen in de antwoorden op query's, kan de database in zekere zin het risico op het geven van een fout antwoord verminderen ten koste van wat twijfel in de goede antwoorden. Zo'n antwoord is doorgaans dichter bij de waarheid. En op deze manier kun je effectief de kwaliteit van antwoorden op query's meten op soortgelijke wijze als in de information retrieval.

Effectief omgaan met imperfecte informatie

Met behulp van een onzekere database is het mogelijk om effectiever om te gaan met imperfecte informatie. De volgende mogelijkheden hiervoor wil ik er graag uitlichten:

- Bij conflicterende of ambigue gegevens kun je normaal gesproken twee dingen doen: wachten tot de kwestie duidelijk wordt voordat de data in de database worden gezet, of alleen de meest waarschijnlijke situatie opslaan met alle risico's van dien. Met een onzekere database kun je direct de actuele stand van zaken opslaan en op een 'best effort' manier gebruiken zonder wachten en minimaal risico;
- Bedrijfsregels en integriteitsregels kunnen worden gebruikt om gegevens 'in twijfel te trekken'. De 'repair key' is een voorbeeld hiervan: het trekt die data in twijfel waarvoor de key niet uniek is. Ook gegevens uit een referentiebron kunnen gemakkelijk worden gebruikt om niet alleen gegevens te controleren, maar om het resultaat daarvan direct toe te passen, wat de antwoorden bij alle gebruikers gelijk verhoogt in kwaliteit (dichter bij de waarheid brengt);
- Er kan een beleid worden gehanteerd dat feedback van gebruikers gelijk (mogelijk automatisch) kan worden verwerkt zonder die gebruikers volledig te vertrouwen: men kan namelijk eenvoudig alleen de waarschijnlijkheid van iets verhogen;
- Met behulp van een steekproef voor een paar query's kun je een redelijke indicatie krijgen van de kwaliteit van de gegevens en daarop eventuele opschoningsacties baseren;
- Men zegt vaak heel eenvoudig "laten we database X en Y samenvoegen" of "laten we de data verrijken met data van website Z". Vanwege verschillen in conventies, structuren, aannames en vanwege fouten van diverse aard blijkt dit in de praktijk een lastige en langdurige klus. Met behulp van een onzekere database hoeft men niet te wachten totdat de hele klus klaar is en alle problemen zijn opgelost. Men kan effectief een moment kiezen waarop de kwaliteit van de data goed genoeg is en de zaak veel eerder in productie nemen. Kwaliteitscontroles, opschoningsacties en user feedback kunnen parallel aan het gebruik de gegevens verder verbeteren. In mijn eigen onderzoek heb ik voor een gegevensverrijkingstaak met gegevens van het web, met daarin veel entity resolution problemen, aangetoond dat men op deze manier in een fractie van de tijd toch in productie kan gaan.[5] En dat bovendien alleen user feedback al zeer effectief kan zijn in het oplossen van resterende onvolkomenheden in de gegevens [KK09].

Ken Orr heeft ooit gezegd 'the only way to truly improve data quality is to increase the use of that data' [Orr98]. Bovenstaande vier punten zijn duidelijk bevorderlijk voor de 'use of data', omdat op deze manier opschoningsacties en kwaliteitscontroles hand in hand gaan met het gebruik van de data in plaats van geïsoleerd van elkaar.

Conclusie

Onzekere databases betreffen een nieuwe databasetechnologie waarnaar de afgelopen jaren veel onderzoek gedaan is. Een onzekere database maakt het mogelijk om gegevens met veel onvolkomenheden toch op een eenvoudige databasemanier op te slaan, te bewerken en te queryen. In dit artikel wordt specifiek ingegaan op toepassing rondom het effectiever omgaan met gegevenskwaliteit in de context van enterprise- en webdata. Meer weten? Het boek 'Managing and Mining Uncertain Data' [AC09] geeft een goed overzicht van de huidige stand van zaken in het databaseonderzoek rondom onzekere databases.

Referenties

[AC09] Managing and Mining Uncertain Data, Series: Advances in Database Systems, Vol. 35, Aggarwal, Charu C. (Ed.), 2009, XXII, 494 pages. ISBN: 978-0-387-09689-6.

[KK09] van Keulen, M., de Keijzer, A. (2009) Qualitative Effects of Knowledge Rules and User Feedback in Probabilistic Data Integration. The VLDB Journal, 18 (5). pp. 1191-1217. ISSN 1066-8888.

[Orr98] Ken Orr. Data quality and systems theory. Communications of the ACM, 41(2):66–71, 1998.

Noten

1. *http://www.cs.cornell.edu/bigreddata/maybms*
2. *http://infolab.stanford.edu/trio*
3. *http://orion.cs.purdue.edu*
4. *http://mystiq.cs.washington.edu*
5. *Entity resolution wordt ook wel 'record linkage' of 'deduplicatie' genoemd.*

Maurice van Keulen

Dr. Ir. M. van Keulen (m.vankeulen@utwente.nl) is Assistant Professor Data Management Technology bij Universiteit Twente.

Update

Waarom wil SAP Sybase?

SAP-woordvoerder Günther Gaugler ziet een sterke synergie tussen beide bedrijven. SAP is sterk in business- en analytische software, Sybase in datamanagement en mobiele infrastructuur. Vooral dat laatste heeft de interesse van SAP: "Met de acquisitie van Sybase krijgen we toegang tot meer dan vier miljard mobiele gebruikers. Ik heb het dan over iAnywhere, een onderdeel van Sybase. Dat biedt ons een geweldige kans om snel te groeien." Gaugler geeft aan dat SAP zijn pijlen op drie targets richt. "Er is de *on-premise wereld*. Daar zijn we heel groot. We groeien momenteel snel in de *in-memory* wereld. En we willen zeker leider worden in de *on-device* wereld, de mobiele wereld."

In-memory technologie gaat volgens SAP een grote rol spelen in de wereld van realtime analytics. "De gebruikers willen geen uren meer op analyses wachten. De combinatie van onze *in-memory* technologie met de ervaring in de databasewereld van Sybase geeft dat een enorme boost."

SAP krijgt na de overname de beschikking over een eigen database. "Dat is mooi, maar we doen niet mee aan de *stack-war*," aldus Gaugler. "SAP blijft beschikbaar voor gebruikers van alle platformen."

Beide bedrijven werken al jaren samen en vormen volgens Gaugler een goede culturele match. "Bovendien hebben we ervaring met het integreren van grote bedrijven," zegt hij, verwijzend naar de acquisitie van BusinessObjects. "Daar verwachten we geen problemen mee."

Informix gratis downloaden

In een poging open source en Microsoft een slag toe te brengen, heeft IBM besloten tot een opmerkelijke strategie waar het de bekende database Informix betreft. De bestaande versies van Informix zijn hergegroepeerd tot zes producten: drie versies, Small, Medium en Large, voor twee platforms, Windows/Mac en Linux/Unix.

De 'small' en 'medium' versie, genaamd Innovator-C, is gratis downloadable voor het Windows/Mac-platform tot 16 GB memory en 4 sockets of 16 cores; voor het Linux/Unix-platform is dat tot 2 GB memory en 1 socket.

"IBM heeft besloten een tweetal trajecten in te gaan," zegt Bertino Melissen, general manager van Informa, premier business partner van IBM. "Ten eerste wil men nieuwe Informix-gebruikers creëren, ten tweede wil men bestaande Informix-gebruikers tevreden houden. Daartoe heeft IBM alle enterprise-versies van Informix met alle opties en toegevoegde producten gebundeld en omgezet in twee producten. Alle gebruikers van wat nu de Informix Workgroup Edition is, worden automatisch gemigreerd naar de Growth Edition en daar krijgen ze dan alle opties, die in het verleden betaald moesten worden, gratis bij. De klant krijgt dus plotseling veel meer waar voor zijn geld."

Op dezelfde manier wordt de Enterprise Edition de Ultimate Edition.

"Het is een vrij aggressieve strategie," beaamt Melissen. "In het begin zal het zeker minder geld in het laatje opbrengen, maar als ik bijvoorbeeld in de Benelux het marktaandeel met een factor tien kan vergroten komen die revenuen over een jaar of drie toch terug."

Melissen verwacht er heel veel van. "Informix is natuurlijk altijd superieur geweest qua database-technologie. Maar daarmee red je het in de markt niet meer. We kunnen met deze aanpak de databasemarkt een forse draai geven."