Een tweede millenniumprobleem ...?

Mw. drs. V.C. Six RE

Jaren is gewerkt aan het millenniumprobleem. Miljarden zijn uitgegeven om het probleem op te lossen. Scepsis enerzijds toen er beduidend weinig problemen aan het licht kwamen, triomf anderzijds voor dit goede resultaat. 'Voor de volgende honderd jaar klaar', zo luidde het links en rechts. Maar zijn we wel klaar? Hebben we niet eigenlijk en masse voor heel veel geld een nieuw probleem gecreëerd? Dit artikel gaat in op de onbelichte kant van het millenniumprobleem: we hebben het probleem nog niet opgelost.

Het millenniumprobleem

Het millenniumprobleem is ontstaan door het gebruik van zes posities voor een datum in plaats van acht. Of wellicht iets nauwkeuriger: door het gebruik van twee posities voor een jaartal in plaats van vier. Voor het feit dat van oudsher niet wordt gewerkt met acht posities zijn verschillende redenen aan te geven, waarop later zal worden ingegaan.
Feit is dat in de jaren negentig vrijwel alle programmatuur werkte met datumvelden van zes posities en dat de eeuwaanduiding niet werd vastgelegd in de database. Zo was '98' altijd 1998 en werd de '19' erbij gedacht. Dat ging goed totdat het 2000 dreigde te worden en de '00' geïnterpreteerd zou worden als 1900.

Bij het oplossen van dit probleem is het in de eerste plaats gegaan om het verzorgen van een juiste interpretatie van jaartallen zodat op het juiste moment '19' of '20' kon worden toegevoegd.

Grofweg is vervolgens gekozen uit twee oplossingen:
* in alle programmatuur een interpretatie aanbrengen die automatisch een keuze maakt tussen '19' en '20';
* de database uitbreiden met de eeuwaanduiding zodat alle datumvelden acht posities lang zijn en geen interpretatieverschillen meer kunnen optreden.

Eerstgenoemde methode staat bekend als 'windowing'. Aan de tweede methode wordt gerefereerd met databaseconversie.

Windowing

Bij de methode windowing wordt bij gebruik van een jaartal van twee cijfers een stukje interpretatie toegevoegd dat gaat bepalen in welke eeuw dit jaartal ligt. Daartoe wordt overal in de programmatuur waar gebruik wordt gemaakt van een tweecijferige jaaraanduiding een stukje programmatuur toegevoegd dat voor die interpretatie zorg draagt. Het idee is dat als het programma '98' leest, dit wordt geïnterpreteerd als 1998; leest het programma '02', dan volgt de interpretatie 2002. De vraag is dan wanneer er '19' en wanneer '20' wordt toegevoegd. Die interpretatie gebeurt niet volgens een hogere wiskundige formule, maar louter op basis van een vooraf gedefinieerd breekpunt. Een breekpunt van 20 betekent dat invoer van '19' het jaartal 2019 oplevert en invoer van '21' het jaar 1921. Bij een breekpunt van 30 betekent dit, dat invoer van '29' het jaartal 2029 oplevert en invoer van '31' het jaar 1931. Over de keuze van een breekpunt zijn wereldwijd geen afspraken gemaakt. Binnen een organisatie is die afspraak doorgaans wel gemaakt. Veel bedrijven hebben een laag breekpunt gekozen, zoals bijvoorbeeld 30.

Een hoger of lager breekpunt kan tot verschillende fouten leiden. Bij keuze van een hoger breekpunt, zoals 50, is de kans dat de invoer incorrect wordt groter in die gevallen dat historische data worden ingevoerd. Bij een breekpunt van 50 zou de interpretatie als volgt zijn: '49' wordt gezien als 2049 en '51' als 1951. Dat zou kunnen betekenen dat de Tweede Wereldoorlog ineens in de toekomst valt! Ook bij een breekpunt van 30 is het al noodzakelijk goed op te letten voor de gevolgen. Vooral bij geboortejaren en andere vroege gebeurtenissen. Iemand die geboren is in 1921, wiens geboortejaar wordt ingevoerd in een programma dat gebruikmaakt van het breekpunt 30, wordt mogelijk ingevoerd met jaartal '21', hetgeen vervolgens wordt geïnterpreteerd als 2021, als zou betrokkene nog geboren moeten worden.

Databaseconversie

De andere mogelijke oplossing was het aanpassen van de database. Dat zou betekenen dat ieder datumveld uitgebreid zou moeten worden met twee posities voor de eeuwaanduiding. Dat lijkt zo op het eerste gezicht heel simpel, ware het niet dat een dergelijke aanpassing veel meer werk verschaft dan je zou denken. Ieder programma dat met een database werkt waarin een dergelijke datum is opgeslagen, werkt met een vooraf opgestelde definitie over alle gegevens die in die database zitten. Een dergelijke definitie voor een datum luidt bijvoorbeeld:

* datumveld;
* zes lang;
* numeriek (alleen getallen);
* indeling: ddmmjj (ofwel, de laatste twee posities zijn de jaartallen).

Indien nu de lengte van het datumveld wordt uitgebreid tot acht, dan moet ook de definitie in het programma worden aangepast:

* datumveld;
* *acht* lang;
* numeriek (alleen getallen);
* indeling: ddmmjjjj (ofwel, de laatste *vier* posities zijn de jaartallen).

Bovenstaande betekent dat bij een databaseconversie niet alleen de database moet worden aangepast, maar nog steeds ook ieder programma dat met die database werkt.

Bovendien moeten alle programma's die met zo'n database werken allemaal tegelijk worden aangepast. Zodra de lengte van het datumveld is aangepast, moeten ook alle definities in alle programma's zijn aangepast om fouten te voorkomen. Dat vergt een veel grotere coördinatie binnen de organisatie.

Zodra de gebruiker in het heden werkt, verlangt hij dat het programma zelf de eeuwtallen toevoegt aan zijn invoer.

Databaseconversie of windowing: de millenniumkeuze

Wereldwijd is eigenlijk algemeen gekozen voor windowing, met uitzondering wellicht van situaties waarin het aanpassen van de database niet op onoverkomelijke problemen stuitte.

Toch lijkt het zo logisch om die gegevensopslag aan te passen. De oplossing is dan wel complex en daarmee duur, maar dan ben je verder voor altijd van het probleem af. Bij een databaseoplossing weet je immers (bijna) zeker dat zich geen fouten meer zullen voordoen van een bejaarde, geboren in 1921, die ineens een baby wordt, omdat zijn geboortejaar als 2021 wordt geïnterpreteerd. Het uitbreiden van de database lijkt een afdoende oplossing. Eén aspect wordt daarbij echter nooit aangeroerd en dat is de gebruiksvriendelijkheid van programmatuur.

Gebruiksvriendelijkheid

Als reden voor het gebruik van zes posities wordt vaak aangegeven het argument dat in de jaren zestig en zeventig geheugenruimte duur was en zuinig moest worden omgesprongen met die ruimte. Deze opmerking doet vermoeden dat als programmeurs vanaf het begin steeds acht posities hadden gekozen, en dus vier posities voor een jaartal, dat er dan geen probleem zou zijn geweest. Dat is gedeeltelijk waar, maar er is meer onder de zon.

Stel, iemand voert via een invoerscherm gegevens in met jaartallen. Bij die invoer wil de gebruiker zo snel mogelijk kunnen werken en verlangt hij of zij dat '99' genoeg is om 1999 aan te duiden. Het bij iedere invoer herhalen van '19' zal de gebruiker niet weten te waarderen. Zeker niet wanneer dit het lopende jaar is. Iemand die in het jaar 1999 gegevens invoert en '99' invoert verwacht dat de programmatuur slim genoeg is om in te zien dat het 1999 is. Een jaar later, in 2000, wil diezelfde gebruiker dat bij invoer van het getal '00' de programmatuur direct inziet dat het gaat om het jaar 2000. Dat is immers het lopende jaar. De gebruiker zit dus helemaal niet te wachten op vier posities voor een jaartal, maar verwacht dat het programma waarmee hij werkt zelf een interpretatie maakt van het jaartal en liefst zo correct mogelijk.
Bij gegevens in de toekomst of in het verleden is de gebruiker nog wel bereid om een extra '19' of '20' in te voeren, maar zodra de gebruiker in het heden werkt verlangt hij enige mate van intelligentie van zijn programma. Het programma moet dus zelf de eeuwtallen toevoegen aan de invoer van de gebruiker en de gebruiker wil zo min mogelijk worden 'lastiggevallen' met foutieve interpretaties.

Bovenstaande toont aan dat het uitbreiden van de database met een jaartal van vier posities lang, niet vanzelfsprekend tot de gewenste oplossing van het millenniumprobleem zou hebben geleid. Een dergelijke aanpassing van de database had immers direct geresulteerd in geroep om meer gebruiksvriendelijkheid en had nog steeds geleid tot het ontwikkelen van een interpretatieprogramma dat veel weg heeft van de interpretatie die bij windowing wordt gebruikt. De afweging die gemaakt moet worden, behelst dus niet alleen geheugenruimte, maar ook gedrag van gebruikers.

Een nieuw millenniumprobleem kondigt zich reeds aan

Uitgaande van bovenstaande conclusie dat het merendeel der bedrijven gekozen heeft voor windowing en gekozen heeft voor een laag breekpunt, bijvoorbeeld 30, is het slechts een kwestie van wachten op een tweede millenniumprobleem. Zodra in een dergelijk bedrijf een jaartal wordt ingevoerd als '31' dat bedoeld was als 2031, maar geïnterpreteerd wordt als 1931, ontstaat vanzelf weer een interpretatieprobleem, met alle mogelijke gevolgen van dien.

Wanneer dat moment zich aandient is afhankelijk van het bedrijf. De kans dat we op dit moment een jaartal willen invoeren boven de 2030 is doorgaans gering.

Maar die relaxte houding zal niet lang duren. Er zijn programma's in deze wereld die een behoorlijk aantal jaren vooruit kijken. Hoever een programma vooruitkijkt is de horizon van een programma. Een horizon van acht jaar is niet ondenkbeeldig. Dat betekent dat in 2022 voor het eerst wordt gewerkt met het jaar 2030. In dat geval krijgt dat programma in 2023 problemen. Voer je in 2023 het jaartal '31' in dan maakt het programma daar automatisch 1931 van dankzij het breekpunt van 30.

De vraag is ook nog hoe alle programmatuur met 30 zelf omgaat. Is er binnen het bedrijf een duidelijk strikt beleid geweest hoe om te gaan met het breekpunt zelf? Moet 30 vallen onder 1930 of onder 2030? Als dat beleid niet duidelijk is geweest, zal het nieuwe millenniumprobleem zich een jaar eerder laten gelden. Invoer in 2022 inzake het jaar 2030 zal niet eenduidig worden geïnterpreteerd als 2030, maar willekeurig, soms als 1930, soms als 2030.

Ergo: is sprake van een bedrijf dat gekozen heeft voor windowing met een breekpunt van 30 en een programma met een horizon van acht jaar, dan zal dat bedrijf uiterlijk in 2021 stappen moeten ondernemen voor de aanpassing van zijn breekpunt.

Een nieuw probleem met eenzelfde omvang?

De opdracht voor het aanpassen van het breekpunt van bijvoorbeeld 30 naar 50 of 60 is helder. Maar aan die helderheid heeft het bij het millenniumprobleem zelf ook niet gelegen. Het is de uitvoering die steeds toch meer werk kost dan verwacht. Hoeveel werk het kost om het breekpunt aan te passen is afhankelijk van een aantal zaken:

* Is een volledige inventarisatie aanwezig van alle programmatuur?
* Is duidelijk gedocumenteerd waar en hoe het breekpunt is ingebouwd?
* Hoe eenvoudig is de wijze waarop het breekpunt is ingebouwd?

Inventarisatie

Veel tijd is bij het millenniumprobleem gaan zitten in een volledige inventarisatie van alle programmatuur. Op dit moment is een inventarisatie beschikbaar, omdat de millenniuminventarisatie zo recent heeft plaatsgevonden, maar het is de vraag of die up-to-date zal worden gehouden. Is dat niet het geval, dan zal opnieuw gezocht moeten worden naar alle programmatuur in het bedrijf. Het is immers weer van groot belang dat van *alle* programma's het breekpunt wordt aangepast. Een grondige inventarisatie moet deze volledigheid garanderen. Bovendien zal de horizon van ieder programma opnieuw moeten worden vastgesteld, opdat kan worden bepaald welk programma het eerst moet worden aangepast.

Documentatie

Na het vaststellen van een volledige lijst met programmatuur zal per programma moeten worden vastgesteld óf het windowing bevat en zo ja, wáár die windowing in het programma staat beschreven en vooral waar het breekpunt is gedefinieerd. Vooral als windowing meer dan eens in één programma is opgenomen, is het van groot belang dat voor alle gevallen het breekpunt wordt aangepast.
Bij het millenniumprobleem zijn scanners gebouwd om te zoeken naar datumvelden en het gebruik van datumvelden. Nu zal gezocht moeten worden naar windowing en breekpunten, tenzij duidelijk in de programmadocumentatie is vastgelegd waar de windowing en de breekpunten op welke wijze te vinden zijn. Niet uitgesloten moet worden dat in het jaar 2020 breekpuntscanners op de markt verschijnen.

Wijze van inbouw

Bij windowing is feitelijk sprake van een stukje programmatuur dat een beslissing neemt tussen 19 of 20. We noemen dit het windowalgoritme. Het breekpunt is een bestanddeel van dit algoritme. Bij de noodzakelijke aanpassing hoeft als het goed is alleen het breekpunt te worden gewijzigd; het windowalgoritme kan ongewijzigd blijven.

Bij het oplossen van het millenniumprobleem had een programmeur de keuze uit twee mogelijkheden:
1. Breekpunt en algoritme staan bij elkaar.
2. Breekpunt en algoritme staan separaat van elkaar, waarbij het breekpunt centraal staat gedefinieerd.

In het eerste geval zullen alle stukjes windowalgoritme moeten worden opgezocht en moet in ieder stukje het breekpunt worden aangepast. Dit kost veel tijd, geeft kans op fouten, zal leiden tot de ontwikkeling van een scanner en behoeft bovendien een serieuze testronde. Als gekozen is voor een centraal gedefinieerd breekpunt hoeft per programma slechts die centrale definitie te worden opgezocht om het breekpunt aan te passen. Met een duidelijk stuk documentatie verlicht dat de klus aanmerkelijk, is de kans op fouten gering en blijven de kosten beperkt.

Ook hiervoor geldt weer de vraag in hoeverre binnen een bedrijf bij het aanpassen met het oog op het millenniumprobleem een richtlijn werd uitgevaardigd dat het breekpunt centraal moest worden gedefinieerd.

Ieder bedrijf zijn eigen millenniumprobleem

De beslissing over de waarde van een breekpunt is voor ieder bedrijf verschillend geweest. In het algemeen is de keuze voor een breekpunt binnen een bedrijf wel gestandaardiseerd, maar wereldwijd zeker niet. Wel zie je dat bedrijven doorgaans een decennium hebben gekozen; breekpunten als 35 kom je niet tegen.

Naast een vrije keuze voor het breekpunt stond het de bedrijven vrij op welke wijze windowing vervolgens werd geprogrammeerd. Hopelijk heeft een bedrijf hierin een gestandaardiseerde oplossing gekozen. Dat is echter niet per se noodzakelijk.

In deze twee beslissingen ligt de vraag wanneer een bedrijf wordt geconfronteerd met zijn 'tweede millenniumprobleem' en hoe omvangrijk vervolgens de klus zal zijn om dat probleem op te lossen.

Mw. drs. V.C. Six RE is na elf jaar EDP-auditor te zijn geweest bij KPMG thans ruim vier jaar werkzaam bij de KLM als IT- en operational auditor. Sinds 1997 heeft zij zich intensief beziggehouden met het millenniumprobleem in de luchtvaartindustrie. In de tachtiger jaren was zij betrokken bij een project waarbij bewust het millenniumprobleem werd omzeild.

Waarom werd het millenniumprobleem niet eerder aangepakt?

De programmeur pakte het probleem niet aan

Het is onterecht te beweren dat IT-specialisten in de jaren zeventig, tachtig en negentig niet bekend waren met het millenniumprobleem. Misschien hadden ze het niet allen helder op hun netvlies, maar er was wel degelijk bekendheid met het probleem. Niemand echter die toen geïnteresseerd was. Een programmeur die een nieuw stuk programma bouwde deed dat enkel en alleen op verzoek van een gebruiker en moest dus voldoen aan de kwaliteitseisen die de gebruiker stelde en het budget dat daarvoor werd afgegeven. Om het omzeilen van een toekomstig millenniumprobleem werd niet gevraagd.

Zoals in het begin van dit artikel aangegeven zou iedere oplossing van het millenniumprobleem, ook in een vroegtijdig stadium, geleid hebben tot extra werk en dus extra kosten. Een programmeur die het millenniumprobleem toentertijd wilde omzeilen, werd dus hoe dan ook geconfronteerd met extra kosten die niet verhaalbaar waren op de gebruiker. Een programma zonder millenniumprobleem was dus duurder en niemand was bereid voor die duurdere versie te betalen. Hoewel uiteraard een aantal programmeurs zich het millenniumprobleem niet gerealiseerd zal hebben in die dagen, was het in sommige gevallen zeker een bewuste keuze programmatuur te maken die niet millenniumproof was. Mede ingegeven vanuit de gedachte dat de programmatuur 2000 niet zou halen en tegen die tijd inmiddels vervangen zou zijn.

> Ook vandaag zal niemand bereid worden gevonden tot het aanpassen van een database of slechte window.

De programmatuur wordt vervangen

Voor een aantal programma's die in de jaren zestig, zeventig en tachtig werden gebouwd, werd gesteld dat die inmiddels zouden zijn vervangen in het jaar 2000. Dat bleek niet het geval. Veel programma's worden niet vervangen, maar aangepast. Een programma is en blijft een hele investering niet alleen in geld, maar ook in het implementeren ervan in de organisatie. Het is vaak eenvoudiger, of er wordt gedacht dat het eenvoudiger is, om het oude programma als basis te (blijven) gebruiken, dan om een geheel nieuw programma in te voeren. In dat geval blijft de oude database bestaan en blijft een datum met zes posities in gebruik.

Het kan ook zijn dat wel een nieuw programma wordt gemaakt, maar dat de oude database in gebruik blijft. In dat geval zal ervoor gekozen worden de database zoveel mogelijk te handhaven zoals die was, omdat dat de minste kosten met zich meebrengt.

Derhalve zullen we ook in de toekomst nog heel lang databases houden met datumvelden van zes posities en programma's die met windowing werken.

Het millenniumprobleem wordt niet onderkend

Uiteraard is er in heel veel gevallen sprake geweest van een miskend probleem. Hoe kon het anders dat er anno 1998 nog nieuwe programmatuur op de markt kwam die geen rekening hield met het millenniumprobleem terwijl er inmiddels al miljoenen werden uitgegeven aan het oplossen ervan?

'Na mij de zondvloed'

Vandaag weet iedereen nog van het millenniumprobleem, maar ook vandaag zal niemand bereid gevonden worden tot het aanpassen van een database of het aanpassen van een slechte window. 'Na mij de zondvloed' denkt menig manager en beperkt zich vervolgens weer tot uitgaven met een duidelijke inkomstenverwachting. En daar vallen het aanpassen van een database en windowing nou eenmaal niet onder.

Conclusie

Gejuich en scepsis voor een wereldwijd probleem dat zonder veel ellende is afgewenteld. Gejuich voor het feit dat we daar voorlopig vanaf zijn. Scepsis vanwege die eenvoud waarmee wordt gedacht dat daarmee de kous af is. Bovenstaande toont aan dat we over zo'n twintig jaar alweer op ons volgend millenniumprobleem(pje) afstevenen, want:

* databases worden niet aangepast;
* programma's werken met windowing en zullen dat voorlopig blijven doen;
* en niemand is nu geïnteresseerd in het oplossen van een probleem dat zich vanaf ongeveer 2020 openbaart. Dat waren we in 1980 toch ook niet?

In de twintiger jaren van de eenentwintigste eeuw zullen we weer voor het eerst worden geconfronteerd met een nieuw millenniumprobleem ergens op de wereld. Vanaf dat moment zal er ieder jaar ergens op de wereld een programma zijn dat een horizon heeft dat door zijn breekpunt heen loopt en zo een potentieel gevaar vormt. Steeds ergens anders, maar niemand weet welk bedrijf, welk programma, op welk moment een probleem heeft. Dat is dan het nieuwe verrassende aspect van dit millenniumprobleem 'nieuwe stijl'. Het zal ook geen wereldwijd probleem zijn, maar wel een wereldwijd gevaar dat ieder jaar rondwaart.

Bedrijven kunnen die problemen vóór zijn door nu te zorgen voor duidelijke documentatie en draaiboeken over het waar, wanneer en hoe aanpassen van hun programmatuur, opgemaakt door de programmeurs van vandaag die de kennis nog hebben van het millenniumprobleem, omdat ze het immers net zelf hebben opgelost.