CUG handleiding

door Thijs Verhagen

CUG handleiding

door Thijs Verhagen

Deze tekst is bedoeld als een leidraad bij CUG (v/h 'configurable user generator'). In het vervolg wordt uitgelegd waar CUG voor dient en wordt beschreven hoe je een eenvoudige testsituatie opbouwt en gebruikt.

Met dank aan de medewerkers van IBO (Hogeschool 's-Hertogenbosch). Commentaar van Frank Dijkstra was erg waardevol, evenals de gesprekken met Piet Brekelmans.

Inhoudsopgave

Lijst van figuren

CUG als hulpmiddel voor systeem-administratie

Inleiding

Iedere organisatie heeft tegenwoordig persoonlijke email voor alle medewerkers. Achter deze eenvoudige zin gaan voor menig systeembeheerder heel wat hoofdbrekens schuil. Hoe komt hij er achter wie alle medewerkers zijn? Hoe vaak wil hij het gebruikersbestand bijwerken? Moet een nieuwe medewerker ook op andere systemen worden aangemaakt? etc. Meestal is het afdoende om een enkelvoudige aanvraag zorgvuldig te verwerken, maar naarmate het aantal gebruikers en het aantal variaties op het systeem groeit, wordt het beheren lastiger. Welke stappen zijn er nodig om een gebruiker netjes af te voeren? Moeten er misschien gegevens gewijzigd worden? etc.

De systeembeheerder zorgt voor primaire toegang tot de IT-voorzieningen. De *medewerker* wordt dan *gebruiker* van mail-, file-, print- en applicatiediensten. Daarnaast is ook binnen applicaties steeds vaker sprake van persoonsgebonden toegang. Het is de taak van een applicatiebeheerder om nieuwe personen met de juiste rechten als *user* aan te melden. Hiervoor heeft hij grofweg dezelfde gegevens nodig als de systeembeheerder, maar dan voor een speciale groep.

In de meeste organisaties is een afdeling personeelszaken verantwoordelijk voor de juistheid van deze gegevens. Voor ander soort gebruikers kan deze registratie ook door andere afdelingen gedaan worden; bijvoorbeeld een verkoop-afdeling in een e-business opzet, of een afdeling studentregistratie op een school. Het is in ieder geval zo dat het beheer van mail- en applicatie-accounts gekoppeld is aan een registratie van personen. Een systeem- of applicatie-beheerder maakt het niet veel uit wie deze registratie doet, als het maar duidelijk is wat met welke gegevens moet gebeuren. Vaak is het ook wenselijk dat gegevens in een bepaald formaat worden aangeleverd.

Ideaal gesproken verloopt het registreren van personen en het aanmaken/verwijderen van systeem-toegang in een vloeiend proces. In de praktijk is dit proces echter lastig in te richten. De verschillende partijen hebben sterk uiteenlopende werkwijzen en de taken waar het om gaat (registratie-nummers kopiëren e.d.) horen vaak tot de minst interessante voor de betrokken medewerkers. Een andere complicerende factor is het voortdurend uitbreiden van applicaties die gebruik maken van persoonsgebonden toegang. Dit geeft aanleiding tot allerlei sub-protocollen en vage aftakkingen (' ... als dat gedaan is, moet je wachten tot je hoort van die en die voordat je verder gaat met...').

Wat zo simpel leek: iedereen een persoonlijke systeem-account geven, is gaandeweg onoverzichtelijk geworden.

CUG dient als hulpmiddel om deze complexiteit beheersbaar te houden. Het doet dit door:

1. De verhouding tussen een account-bestand en een personen-bestand te reguleren

   CUG maakt het mogelijk periodiek een vergelijking van deze bestanden te doen. Als de bestanden compleet worden aangeboden komt dan in de CUG-database een actuele momentopname te staan van welke persoon bij welke gebruiker hoort. Ook zijn er CREATE/DELETE/UPDATE sets aangemaakt.

   Het is de bedoeling hiermee het account-bestand te wijzigen. Het personen-bestand leidt het account-bestand. Het account-bestand waar het hier om gaat, is het primaire account-bestand (*enterprise directory*). Het kan op zijn beurt weer andere (applicatie) account-bestanden leiden. CUG is gemaakt voor een omgeving met een NDS als primair account-bestand.

2. Zinvolle selecties mogelijk te maken op de gegevens uit deze bestanden.

   Denk bijvoorbeeld aan: 'alle nieuwe gebruikers uit die en die afdeling'; 'alle te verwijderen gebruikers voor die en die container' etc.

3. Flexibele rapportages te bieden op deze selecties.

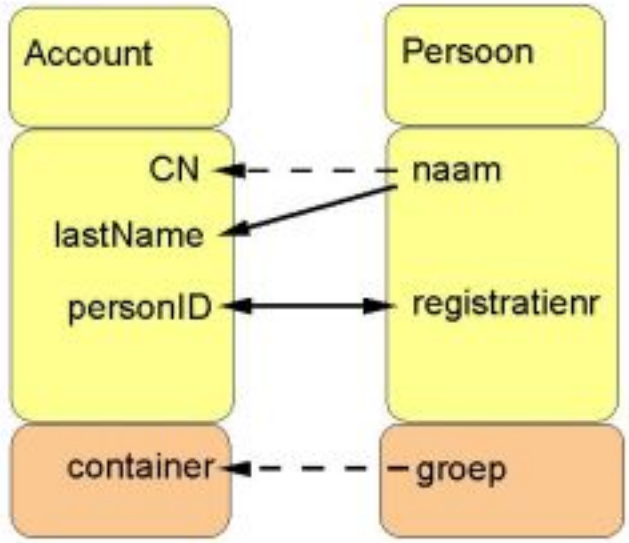
   Hiermee worden onder andere tekst-tabellen met verschillende kolomvolgorden bedoeld.

Belangrijk

CUG gaat er van uit dat het gemakkelijk is om een totaal-overzicht op te vragen van de accounts en de persoonsregistraties.

CUG is ontwikkeld met het idee dat het niet mogelijk is de complexiteit van dit proces geheel en al op te heffen (*no silver bullet*). Het mag echter niet zo zijn dat het opvragen van consistente gegevens uit beide bestanden een struikelblok wordt. Men moet zich kunnen concentreren op het vaststellen van protocollen cq maken van afspraken. CUG levert een solide koppeling tussen persoon en account die dit gemakkelijker maakt.

Figuur 1. Koppeling persoon / account



In dit diagram zijn de basis-attributen van beide entiteiten afgebeeld. De koppeling is aangegeven met pijlen; een stippellijn geeft aan dat er een regel gehanteerd wordt voor de koppeling; bijvoorbeeld de regel 'eerste twee voorletters en vier letters van de achternaam' om een CN[1] te maken van de naam van een persoon.

CUG is bedoeld als bouwsteen om een proces te ondersteunen. CUG wordt geconfigureerd en kan dan zonder verdere tussenkomst van een gebruiker de momentopname periodiek bijwerken. Om die reden is er geen prioriteit gegeven aan een grafische user interface bij CUG. Voor statische

[1] CN is een afkorting voor *common name*. Hiermee wordt de 1ste systeemnaam van een persoon bedoeld, ook wel de loginnaam of userid. In een NDS is dit een woord zonder spaties.

Een DN (*distinguished name*) is een concatenatie van de CN en de container-naam. Een DN is de unieke aanduiding van elke user in een directory.

instellingen wordt gebruik gemaakt van java properties-files.

Meer dynamische situatie-gebonden instellingen zoals regels die gehanteerd worden voor de inrichting van het systeem (de NDS) en de manier waarop CUG de input-data verwerkt dienen per organisatie in java gecodeerd te worden.

Om flexibel te blijven moet CUG open en uitbreidbaar zijn, met name aan de rapportagekant. In eerste instantie gebruik je hiervoor weer java, CUG is dan zoiets als een gereedschapskist voor accountbeheer.

Doelgroep

Bij het in het groot beheren van mail- en applicatie-accounts zijn verschillende partijen betrokken. Zoals we gezien hebben kunnen met name systeem- en applicatiebeheerders profiteren van de mogelijkheden van CUG. Voor succesvol gebruik is onderlinge coördinatie van groot belang. Het is zeker nuttig de invoering van CUG - of enig ander hulpmiddel bij het centraal beheren van accounts - in een projectvorm te doen. De projectleider zal ook begrip moeten hebben van CUG.

In deze tekst wordt een demo gepresenteerd die gebruikt kan worden om de mogelijkheden CUG te verkennen. Hoewel de kennismaking zo is opgebouwd dat iedereen met een basis computervaardigheid hem kan doorlopen, kan het zijn dat men niet in alles geïnteresseerd is.

- Niet-gebruikers van CUG, dat wil zeggen personen die alleen coördinerende rol hebben, hebben na het lezen van dit voorwoord genoeg kennis. (In de toekomst zal een implementatievoorbeeld nader uitgewerkt worden).
- Systeembeheerders, dat wil zeggen beheerders van het primaire account-bestand, doen er goed aan de tekst in hoofdstuk 3 te bestuderen. Hierin wordt uitgelegd hoe CUG bepaalt welke gebruikers aangemaakt of verwijderd moeten worden.
- Voor applicatiebeheerders geldt dit ook, maar voor hen is het bekijken van de rapportagemogelijkheden uiteindelijk interessanter.

CUG is zelf natuurlijk ook een applicatie. Als CUG als hulpmiddel gebruikt wordt, zal er ook iemand het (technisch) beheer op zich moeten nemen. Voor die persoon is deze tekst een gemakkelijk beginpunt.

Hoofdstuk 1. Voorbereiding

Om de demo te doorlopen is een basiskennis van het installeren en gebruiken van java-programma's behulpzaam. Als je weet wat een JRE is en een CLASSPATH, zal deze tekst gemakkelijk te volgen zijn. Het is niet noodzakelijk om de java-code te begrijpen - begrijpen hoe de meegeleverde batch-files in elkaar zitten is belangrijker. Voor meer informatie over het gebruik van java-programma's begin bij www.java.sun.com [http://www.java.sun.com].

Kennis van SQL en het werken met ODBC is ook erg nuttig. Om de data aanschouwelijk te houden is het voorbeeld gemaakt met een MS Access-bestand als doel. Als Access niet beschikbaar is moet een andere ODBC-driver gebruikt worden, bv die voor tekst-bestanden. (Er is ook een installatie op MySQL getest.)

In de demo wordt veel gewerkt met java properties-bestanden. Die zijn ongeveer hetzelfde opgebouwd als Windows ini-bestanden, maar dan zonder secties. Bij het aanpassen van deze bestanden moet je er rekening mee houden dat back-slashes in padnamen verdubbeld moeten worden. In Java 2 is het ook mogelijk '/' te gebruiken ipv '\' in padnamen.

Belangrijk

Alle sleutelnamen in java-properties files zijn hoofdlettergevoelig.

De instructies zijn opzettelijk compact gehouden. Als er bv staat 'wijzig ... in dit bestand' wordt het openen en naderhand opslaan van het bestand niet verder uitgewerkt. Ook mag de lezer zelf bepalen met welk gereedschap de tekstbestanden worden bewerkt. In Aanhangsel A. Typische demo-sessie staan een paar tips voor het configureren en testen van CUG.

De CUG-demo bestaat uit de volgende onderdelen:

- java-classes

  Een gedeelte van de site-configuratie van CUG wordt gedaan in java. in `java/src` staan 3 java bronbestanden, in `java/build` de gecompileerde classes.

- data-bestanden

  In `importdata` staan enkele tekstbestanden die worden gebruikt als databron. `ndscur.data` bevat het gehele overzicht van de nds; persoonsgegevens worden gehaald uit `person.data` en `customer.data`. De bestanden met de extensie '.in' bevatten de oorspronkelijke demo-data.

- java bibliotheken

  `cug.jar` bevat de cug-classes; `log4j-core.jar` bevat een logging package afkomstig van het apache jakarta [http://www.apache.org/jakarta] project

- een set batch-files en configuratie-bestanden

  In de map `run` van de demo-installatie bevinden batch-files die op de demo van toepassing zijn en tekst-bestanden die het runtime gedrag van CUG beinvloeden.

Hoofdstuk 2. Installatie

Om de CUG-demo te gebruiken moet een werkstation aan de volgende voorwaarden voldoen:

- ODBC is beschikbaar. Bij een standaard installatie van Windows is dit zelden een probleem.
- Er is een Java 2 runtime environment (JRE) beschikbaar.
- Er is unzip gereedschap en een teksteditor voorhanden.

De batch-files in de demo gaan er van uit dat `java.exe` te vinden is via het DOS-path.

Tip

Ook op een novell-server (versie 5 of hoger) is een `java.exe` voor Java 2 vaak openbaar beschikbaar (public).

Installatie procedure

De installatie procedure heeft 4 stappen:

1. Installatie van de odbc databron
2. Uitpakken en nabewerken van het demo-programma
3. Inrichten van de database
4. Controle

1. **Installeer een odbc-datasource**

   a. Kies: Start, Instellingen, Configuratiescherm, ODBCGegevensbronnen; tab Systeem-DSN, Toevoegen; selecteer de MS Access Driver en geef Enter

      Geef de data source een naam; bv **`cug_sample`**. Onthou deze naam, je hebt hem bij het installeren later weer nodig. Voer eventueel een beschrijving in.

   b. Druk op de knop Create om een Access-bestand aan te maken. Er verschijnt een bestandsdialoog; selecteer een map en geef het bestand een naam. Als alles goed gaat, komt hiervan een melding. Onthou ook de (pad) naam van het Access-bestand.

2. **Installeer het voorbeeld**

   Unzip de bestanden in `cugtutorial.zip` in een map naar keuze. In die map komen de eerder beschreven bestanden te staan.

   In de map `run` staat een bestand `cugprogram.ini`. Met dit bestand wordt de omgeving van CUG ingesteld. In de praktijk zal dit bestand niet vaak gewijzigd worden nadat het eenmaal ingesteld is. Voor de werking van de demo is het van belang de data source name (dsn) aan te passen.

a. Open `cugprogram.ini` met een teksteditor

b. Pas de regel met 'dsnTarget' aan, zodat achter het =-teken de naam van de datasource staat die bij installatie is aangemaakt (bv **cug_sample**)

Opmerking

Normaal gesproken zul je ook de waarde voor 'outputDir' aanpassen. De batch-files in de demo gaan er echter van uit dat alle output in de map `out` terecht komt. Dit is met name bij het simuleren van een systeem-update (**do_import**) van belang.

terug naar overzicht

3. **Installeer de database**

   Bij CUG hoort een installatie module die de database inricht waar 'dsnTarget' naar verwijst.

   Ga naar de directory `run` en start **install** op.

Tip

Als je in `cugprogram.ini` 'mySql' op true zet en je draait **install** wordt in een sql-script `cugcreate.sql` aangemaakt, dat kan dienen om de database in te richten voor MySQL.

terug naar overzicht

4. **Controle**

   Als je nu het achterliggende access-bestand opent, zie je dat de volgende tabellen zijn aangemaakt:

Het programma is nu klaar voor gebruik.

CUG tabellen

tabellen met nds-users

NCREATE	hierin komen nds-users die aangemaakt moeten worden
NDELETE	hierin komen nds-users die (mogelijk) verwijderd kunnen worden
NDSEXPORT	hierin komt het resultaat van de nds import
NUPDATE	hierin komen nds-users waar iets aan gewijzigd moet worden

Deze tabellen hebben alle vier dezelfde structuur en worden hier ook wel N-tabellen genoemd.

tabellen met persoonsgegevens

PERSON	hierin komen de geïmporteerde persoonsgegevens
PTREE	hierin komt de afdelingen-structuur zoals die met de persoonsgegevens

	wordt geïmporteerd
PTREEATTS	hierin eventueel extra kenmerken van een afdeling
PTREEMEMBER	hierin komt de koppeling tussen persoon en groep

CUG stuur-tabel

SysGroepContainer	Deze tabel bevat de brug tussen afdelingen en nds. Bij een afdeling kan een nds-container opgegeven worden. Via deze tabel weet CUG waar een gebruiker thuis hoort. De waarden in deze tabel worden bij installatie ingelezen uit het bestand `install_initgroepcontainer.properties` in de `run`-directory.

Hoofdstuk 3. Werking en gebruik

Inleiding

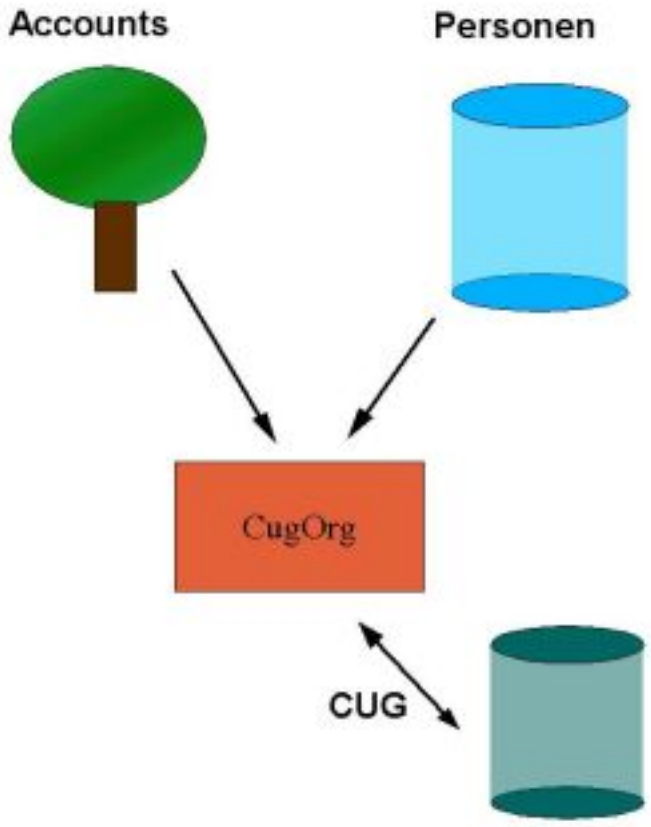
Kort gezegd gebruik je CUG om persoons- en accountgegevens te verwerken tot 'persoon + account gegevens'.

Hiervoor gebruikt CUG momentopnamen van beide gegevenssets. Deze momentopnamen (*snapshots*) worden in de database opgeslagen (NDSEXPORT en PERSON).

Vervolgens kunnen deze gegevens vergeleken worden. Primair gaat dat via een sleutelveld (PERSONID) dat in beide gegevenssets voorkomt. Deze vergelijking heeft een vulling van de CUG N-tabellen als resultaat. Dit resultaat is uiteraard ook een momentopname.

Onderstaand diagram geeft dit weer.

Figuur 3.1. CUG import



Het onderdeel CugOrg is het situatie-gebonden gedeelte van CUG, bestaande uit java-classes en configuratie-bestanden.

CUG gaat er steeds van uit dat de persoons-import volledig en accuraat is. Als dit zo is heb je een hulpmiddel om het beheer van systeem-accounts te laten leiden door de registratie van personen door een adminstratieve afdeling.

Na de vergelijking is het mogelijk de inhoud van de N-tabellen te gebruiken als basis voor CUG-rapporten. Het idee is dat het gemakkelijk moet zijn om tussen twee update momenten verschillende rapporten consistent te houden (dezelfde initiële wachtwoorden, dezelfde nieuwe gebruikers etc).

Het is natuurlijk ook mogelijk de tabellen rechtstreeks met queries te benaderen.

Hieronder wordt de CUG update slag als een 3 afzonderlijke stappen bekeken. In de demo is het mogelijk wijzigingen in de nds- en persoonsbestanden snel te verwerken met behulp van **CUGDaily**

Het feit dat dit losse stappen zijn, maakt het mogelijk ze afzonderlijk te *schedulen*. Je kunt dan bv om 04:00 een nds-import doen, om 05:00 een momentopname van de persoonsgegevens maken en om 06:00 de CUG-vergelijking draaien. Vanaf de ochtend-koffie kun je dan samenhangende rapporten gaan draaien op deze gegevens.

Import van de nds-gegevens

De eerste stap is het importeren van de nds-gegevens.[2]

1. **Bekijk de inhoud van de tabel NDSEXPORT**
2. **Start ndsimport op.**
3. **Bekijk de inhoud van de tabel NDSEXPORT nog eens.**

In de demo wordt het bestand `importdata/ndscur.data` ingelezen. Dat dit zo is, en hoe de regels in dit bestand nds-users voorstellen is vastgelegd in de java class SampleOrg. In de praktijk zullen de gegevens meestal uitgebreider zijn en via jdbc worden ingelezen. Voor de demo is gekozen voor een tekst-bestand. (weer een properties-bestand)

In `ndscur.data` zijn een een paar test- en systeemgebruikers vermeld. Het is niet mogelijk om met een geheel lege nds te beginnen. CUG maakt namelijk geen containers aan. Alleen containers die voorkomen in NDSEXPORT worden als geldig beschouwd.

Bovendien maakt CUG alleen gebruikers aan in de in SysGroepContainer vermelde containers.

In de oorspronkelijke `ndscur.data` (zie `importdata/ndscur.in`) zijn ook gebruikers in andere containers opgenomen om te demonstreren dat CUG:

- *alle* bestaande gebruikersnamen in acht neemt bij het aanmaken van een nieuwe gebruikersnaam
- geen gebruikers aanmaakt in containers die *niet* worden vermeld in SysGroepContainer

Alles alles goed is verschijnt de volgende of vergelijkbare output:

[2] Misschien is het beter als het importeren van de nds-gegevens de *tweede* stap zou zijn. Theoretisch gezien is het gunstig om het verschil in tijdstip van maken van de nds-import en aanmaken van gebruikers (verwerken van het hoofd create-rapport) zo klein mogelijk te laten zijn. Het is immers mogelijk dat een ander programma of een beheerder een gebruiker toevoegt, waardoor het mogelijk is dat CUG een niet-unieke userid als geldig gaat beschouwen voor een create-user.

Figuur 3.2. Output ndsimport

```
0 [main] INFO cug  - CUG instellingen uit: cugprogram.ini
50 [main] INFO cug  - DataSource import: _not used_                    ❶
50 [main] INFO cug  - DataSource doel: cug_sample                      ❷
110 [main] INFO cug  - CugResources; huidige CugOrg class: Cug sample Organization
110 [main] INFO cug.Main  - Verwerken nds import
270 [main] INFO cug.Main  - import bevat 7 nds users;                  ❸
270 [main] INFO cug.Main  - met 0 verschillende sleutels (PERSONID)    ❹
270 [main] INFO cug.Main  - en 3 verschillende login namen             ❺
Deleted table: cug_sample.NDSEXPORT                                    ❻
84 items added to: cug_sample.NDSEXPORT
2080 [main] INFO cug.Main  - Einde updaten nds export
2080 [main] INFO cug.Main  - run in ongeveer: 2 seconde
```

❶ De import wordt hier *niet* gedaan vanaf een jdbc databron. (De begin- en eind-underscore wijzen op een CUG defaultwaarde.)
❷ Resultaat wordt weggezet op jdbc databron 'cug_sample'
❸ Er worden 7 users herkend in de import.
❹ Die hebben geen van allen een nummer uit het administratieve systeem.
❺ Deze 7 users hebben 3 verschillende userid's; in de database kun je zien dat dat 'test' 'backup' en 'systemadmin' zijn
❻ Merk op dat er ook regels worden gelogd met een ander formaat (zonder milliseconde sinds start)

Bij elk gebruik van CUG wordt dergelijke logging gedaan. Het formaat waarin dit gebeurt, wat wel en wat niet gelogd wordt, of er ook naar een bestand gelogd wordt etc. is in te stellen door `log4j.properties` aan te passen (zie www.apache.org [http://www.apache.org/jakarta] voor details). Sommige meldingen worden nog rechtstreeks naar de console gedaan.

Import van persoonsgegevens

1. **Bekijk de inhoud van de tabel PERSON**
2. **Start import op.**
3. **Bekijk de tabellen PERSON, PTREE en PTREEMEMBER om een indruk te krijgen van het resultaat van de import.**

CUG zoekt naar persoonsgegevens in de bestanden `importdata/person.data` en `importdata/customer.data`. Dit wordt bepaald door de java class SampleOrg.

De voorstelling in de demo is dat in `person.data` medewerkers van een demo-bedrijf zitten, en in `customers.data` klanten. In de praktijk zullen ook vaak persoonsgegevens uit verschillende bronnen worden gebruikt. Het is de verantwoordelijkheid van de CugOrg (SampleOrg hier) om verschillende bronnen (en -bestandsindelingen en -momenten) samen te voegen.

CUG slaat afdelingsgegevens op in de PTREE-tabellen en persoonsgegevens in PERSON. CUG verwerkt daarbij persoonsnamen zoals in Nederland gebruikelijk.

Voor elke afdeling/groep wordt een code aangemaakt die deze groep in CUG uniek identificeert. Deze codes zijn belangrijk als je selecties van gebruikers wilt maken in CUG. Ze stemmen bv overeen

met de codes in SysGroepContainer.

PTREE ontleent zijn naam aan het feit dat je er een boomstructuren (van arbitraire diepte) in kwijt kunt. Een ptree eindigt in groepen die personen bevatten. De ptree-codes van deze groepen worden gebruikt in PTREEMEMBER om personen in deze groepen te plaatsen. PTREEMEMBER is dus een koppeltabel tussen PTREE en PERSON.

Om weer een overzicht te krijgen van hoe personen in afdelingen zitten zou je de volgende query kunnen uitvoeren in het access bestand.

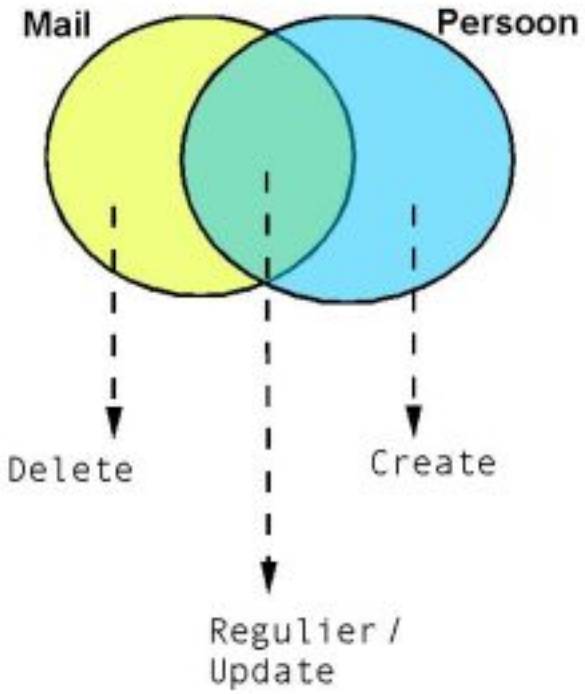
```
select ptree.code, person.*
from ptree, person, ptreemember
where ptreemember.personid = person.personid and ptreemember.groupcode = ptree.code;
```

Kopieer bovenstaande regels met Ctrl-C; ga naar het tab Query; menu Invoegen, Query, Enter; menu Beeld, SQL; plak tekst met Ctrl-V; menu Beeld, Gegevensbladweergave. Ctrl-F4 om query af te sluiten.

CUG vergelijking

In principe is het vergelijken van de persoon- en de accountgegevens niet ingewikkeld, zoals volgende diagram laat zien.

Figuur 3.3. CUG vergelijking



De grootte van het gebied heeft in dit diagram uiteraard geen betekenis. In de praktijk is de intersectie 'regulier/update' (hopelijk) het grootst.

1. **Bekijk de inhoud van de tabellen NCREATE, NDELETE en NUPDATE**
2. **Start update op**
3. **Bekijk bovenstaande tabellen nog eens**

CUG vergelijkt de eerder geïmporteerde gegevens met elkaar. Het resultaat van deze vergelijking wordt in de N-tabellen opgeslagen. In de demo wordt bij de eerste keer dat **update** wordt gebruikt voor alle personen een regel aangemaakt in NCREATE.

Alle personen zijn nu gekoppeld aan een nds-user, alle nds-users waar geen persoon bij is gevonden zijn gekopieerd naar NDELETE. Alleen wijzigingen rechtstreeks op de database kunnen nog verandering brengen in de data waarop straks rapporten worden aangemaakt.

Bij het aanmaken van nieuwe userid's worden een paar regels gehanteerd:

- De context voor een nieuwe user wordt bepaald door de groep/afdeling/ptree waar de bijbehorende persoon in thuis hoort (zie SysGroepContainer)
- De userid is uniek ten opzichte van de nds-export en de overige userid's in NCREATE

Bovenstaande regels behoren tot de specificatie van CUG. De volgende regels zijn aan te passen.

- Een userid wordt gevormd uit de beginletter(s) van de voornaam, de beginletter(s) van het tussenvoegsel en de letters van de achternaam
- Een userid heeft ten hoogste een lengte van 10 tekens
- Het wachtwoord heeft altijd een lengte van 7 tekens

De instellingen voor de demo staan als volgt in het bestand `run/createparams.properties`[3]

```
cug_passwordLen=7
cug_idMaxLen=10
cug_idPattern=F+I+L*
```

Het instellen van de lengte van het wachtwoord en de userid is zo eenvoudig als het eruit ziet.

De opmerking dat een erg kort userid bij een grote groep gebruikers tot problemen kan leiden is bijna overbodig. In het extreme geval: 100 gebruikers met dezelfde achternaam en dezelfde voorletter (bv in een familie stamboom) en een id-lengte van 2, zul je zeker vastlopen.

Zet cug_idMaxLen eventueel op 0 om geen maximumlengte te gebruiken. In dat geval verschijnen alleen cijfers in de userid als meer personen en dezelfde voorletters en dezelfde achternaam hebben.

De waarde bij de sleutel 'cug_idPattern' is een weergave van regels bij het aanmaken van userid's gebaseerd op persoonsnamen. Door hier in te variëren wordt de werking snel duidelijk. De betekenis van de verschillende tekens is als volgt:

1. 'F' staat voor: 1 letter uit de voornaam (first name)
2. 'I' staat voor: 1 letter uit het tussenvoegsel (infix)
3. 'L' staat voor: 1 letter uit de achternaam (lastname)
4. '+' : alle beginletters uit de naam aangeduid met de letter uit het teken *voor* de '+'
5. '*' : alle letters uit de naam aangeduid met de letter uit het teken voor dit '*'

Elk ander teken in 'cug_idPattern' wordt onveranderd (lowercase) meegenomen in de userid.

Bijvoorbeeld: voornaam: 'Jean-Paul'; tussenvoegsel 'van der'; achternaam 'Sterren'

- F+I+L* => jpvdsterren
- F*I+L* => jeanpaulvdsterren
- LLLLFF => sterje
- LLXLFF => stxeje

Merk op dat in de laatste twee voorbeelden (geen gebruik van '+' of '*') cug_idMaxLen minder belangrijk wordt. Hooguit als de volgnummers erg hoog oplopen ('sterje115' ed).

[3] In cugprogram.ini staat een sleutel 'createProps' die instelt naar welk bestand wordt gekeken.

Hoofdstuk 4. Rapporteren

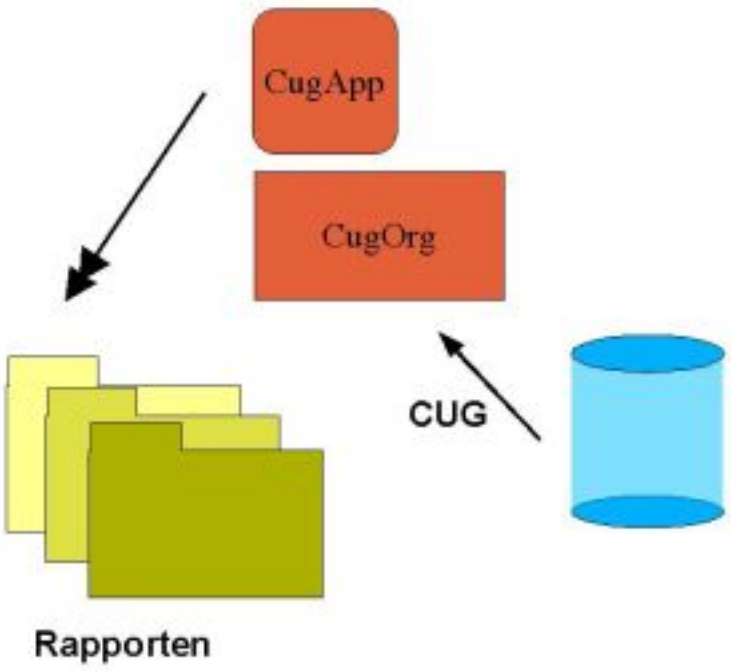
Voor gebruikersbeheer in verschillende systemen worden vaak tekstbestanden gebruikt. Deze bestanden worden in CUG rapporten genoemd.

Omdat CUG de benodigde gegevens heeft opgeslagen in een database, kunnen rapporten in principe worden gemaakt met behulp van queries. Deze queries zijn echter niet altijd even gemakkelijk samen te stellen, vanwege de vele string-bewerkingen die nodig zijn en een moeilijk te voorspellen behoefte aan extra statische waarden (home-directory root; postoffice of bv rol in doelsysteem etc). Daarnaast is soms parallel aan een rapport een configuratie-bestandje nodig dat beschrijft hoe het rapport in elkaar zit.

Daarom heeft CUG voorzieningen om dit soort rapporten aan te maken. Er zijn een aantal standaardrapporten die kunnen dienen als basis voor meer specifieke rapporten. CUG kan ook het aanmaken van rapporten reguleren, bv een datum invoegen in de bestandsnaam van een rapport.

Aansluitend op de import (zie Figuur 3.1. CUG import) kan dit schematisch weergegeven worden.

Figuur 4.1. Rapporteren in CUG



Demo-class SampleApp is een toepassing die laat zien hoe je CUG kunt gebruiken om rapporten aan te maken op de gegevens in de CUG databron. Voor bestandsnamen, selecties en statische waarden maakt SampleApp gebruik van properties-bestanden; met name `sampleapp.properties`.

Create rapport aanmaken

Om te zien hoe je CUG rapporten gebruikt, zullen we eerst het create-rapport aanmaken.

1. **Open het bestand run/sampleapp.properties**
2. **Verwijder het '#' voor de sleutel 'file.ngwImport'**

   Hierdoor verzoek je als het ware een import-bestand aan te maken. Als je het '#' weer voor de regel zet, weet SampleApp dat je geen create-rapport nodig hebt. ('#' betekent: "negeer de rest van de regel" in een java properties-bestand).
3. **Voer SampleApp.bat uit om de rapporten aan te maken.**
4. **Bekijk het resultaat in de map out.**

In het bestand `out/import.data` zul je dan zien dat alleen voor users in 'person' rapporten zijn aangemaakt. Dit is ingesteld dmv de 'ptree.code' in `sampleapp.properties`. De bijbehorende waarde wordt door SampleApp gebruikt om een ptree (afdeling/groep; zie boven) te selecteren.

Rapporten worden vaak per ptree aangemaakt. Meestal zal hiervoor een root ptree worden gekozen ('person' of 'customers' in de demo).

Create rapport verwerken

In een praktijk-situatie zal een hoofd create-rapport verwerkt worden door een systeembeheerder. Het rapport wordt dan geladen in een import-tool zoals UIMPORT. In de demo wordt dit gesimuleerd door het bestand `NDSCUR.DATA` bij te werken. CUG pikt deze gebruikers op bij de volgende update-slag.

1. Start batchfile **do_import.bat** op. Hierdoor wordt de inhoud van `out/import.data` aan `importdata/ndscur.data` toegevoegd.
2. Herhaal de CUG update slag mbv **CUGDaily.bat**

Deze laatste stap is steeds nuttig als je iets hebt gewijzigd in de data-bestanden in de map import.

Werken met verschillende ptree's

Als je de vorige handelingen hebt uitgevoerd, kun je zien dat de tabel NCREATE een stuk geslonken is. Alle personen in 'person.data' zijn verwijderd. Ze hebben immers een account in NDSEXPORT nu. Wat overblijft zijn de users voor de personen in 'customer.data'.

Om deze users op dezelfde manier te verwerken zou het genoeg zijn de ptree-code in `sampleapp.propertieste` wijzigen. Dat zou echter betekenen dat je deze code steeds heen en terug zou moet wijzigen. Misschien zijn er wel andere rapporten nodig voor andere ptree's.

Om die reden is het mogelijk een complete set configuratie-bestanden apart te zetten en CUG te laten kijken naar deze bestanden. Je doet dit door het classpath voor de jre anders in te stellen.

Belangrijk

CUG zoekt niet alleen zijn classes, maar ook zijn configuratie-bestanden (cugprogram.ini, createparams.properties etc.) in het java classpath. Vermijd de situatie waarin meer dan 1 configuratie-bestand met dezelfde functie vindbaar is. Groepeer configuratie-bestanden in een map en voeg een verwijzing naar deze map toe aan het classpath.

De configuratie-bestanden voor het werken met 'customers' staan in de map `run/samplecust`. In **SampleAppCust** is het classpath zo ingesteld, dat gezocht wordt in deze directory. Je ziet dan er iets andere namen gebruikt worden voor de rapporten. Dit is belangrijk bij het controleren van de rapporten.

1. **Bekijk run/samplecust/sampleAppCust.properties.**
2. **Start SampleAppCust op.**
3. **Bekijk het resultaat in de map out (sorteer op datum/tijd).**

Nu ook de rapporten voor ptree 'customers' gemaakt zijn, kan de nds worden bijgewerkt. Voer hiervoor eerst **do_import** uit en dan **CUGDaily** (zie boven).

Als het goed zijn alle N-tabellen nu leeg, omdat voor alle personen een user is geïmporteerd.

CUG variabele rapporten

Als je de instructies op de vorige pagina gevolgd hebt, zag je in `sampleAppCust.properties` het volgende fragment:

```
###
# vrije vorm rapport met ptree selectie
###
cugrapport.header = email;userid;name;personid;formattedsortname
file.cugrapport = cugrapportCust.txt
exist.cugrapport.code = CUSTOMERS.INT
```

Deze drie sleutels worden door SampleApp ingelezen en geïnterpreteerd als: maak voor de afdeling INT in 'customers' een tabel-bestand `cugrapportCust.txt` aan met de kolommen: email, userid etc.

1. **Varieer de waarde voor 'cugrapport.header'; voeg velden toe (zie) of verwijder ze**
2. **Start batch-file SampleAppCust nog eens op en bekijk het resultaat.**

In het andere configuratie-bestand voor sampleapp: `run/sampleapp.properties` staat de volgende tekst:

```
###
# dynamisch geselecteerd rapport
###
extra.rapport.type = cug.report.NDSMapRapport
```

```
extra.rapport.input = ../importdata/personids.lst
file.extra.rapport = extrarapport.txt
```

Ook hier geldt dat de sleutelnamen alleen voor SampleApp betekenis hebben. SampleApp reageert door het rapport NDSMapRapport aan te maken voor users met een sleutel die voorkomt het bestand `personids.lst`. SampleApp maakt gebruik van CUG om via dit configuratie-bestand een rapport te selecteren.

1. **start sampleapp.bat en bekijk het resultaatbestand (file.extra.rapport)**
2. **Wijzig de waarde bij 'extra.rapport.type' in 'cug.report.BBCreateRapport'**

   Pas eventueel ook de doel filenaam aan.
3. **Start sampleapp.bat weer en bekijk het resultaatbestand**

   (Merk op dat het tussenvoegsel niet los voorkomt in de uitvoer.)

Je kunt op deze manier alle rapporten selecteren die een subclass zijn van `cug.report.SimpleNDSMapRapport`.

Hoofdstuk 5. Hoe nu verder

Als de voorbeelden in de vorige hoofdstukken helemaal duidelijk zijn, en je hebt er nog geen genoeg van, zijn er twee richtingen om CUG verder te verkennen. Met de demo-bestanden verder gaan of de java-code in.

Verder testen van de demo

Hiervoor is een goed begrip van hoofdstuk 3 belangrijk.

Daarnaast is het noodzakelijk dat de werking van alle batch-files duidelijk is. Er zijn nog een paar batch-files niet beschreven in de tekst:

- clear.bat wist de inhoud van de N-tabellen;
- update_noclear.bat doet een update van de N-tabellen zonder ze eerst te wissen;
- resetdata kopieert de oorspronkelijke databastanden naar de huidige
- resetdemo.bat kopieert de oorspronkelijke configuratiebestanden naar de huidige

Er twee batch-files die voorbeelden geven van test-scenario's. Voeg evt een 'pause' in waar je de data wilt inspecteren.

1. **Wijzig iets in ndscur.data of person.data**

   NB dubbele personid's in de persoonsgegevens leiden tot data-verlies en inconsistenties.

2. **Voer CUGDaily uit**

3. **Controleer het resultaat in de database of in een rapport verkregen via SampleApp.**

Probeer vervolgens allerlei onregelmatigheden te simuleren. Interessante zaken om te testen met ndscur.data en person.data zijn:

- een persoon toevoegen met dezelfde naam als een bestaande persoon (maar met een ander nummer)
- een persoon verwijderen
- de naam van een persoon wijzigen
- het nummer van een persoon wijzigen
- nog een keer, maar nu met cugprogram.ini: updateLookALikes=true
- de lastname van een nds-user in ndscur.data wijzigen
- een user zonder personid toevoegen aan ndscur.data
- een user met personid toevoegen aan ndscur.data
- etc.

Wijzigingen in SysGroepContainer zijn het gevaarlijkst. Je moet je realiseren dat waarden voor GROEP geldige ptree-codes moeten zijn (zie tabel PTREE). Waarden voor CONTAINER moeten voorkomen in NDSEXPORT. Voldoen je wijzigingen niet aan een van deze twee voorwaarden, dan zullen er run-time fouten optreden en de database kan inconsistent worden.

Om hiervan te herstellen is een **install** in ieder geval afdoende.

Een interessante oefening is het toevoegen van een afdeling. Je moet hiervoor eerst een aantal personen toevogen aan `import/person.data` en deze importeren. Na import kijk je welke code CUG verzonnen heeft voor je afdeling en je voegt een regel toe aan SysGroepContainer met deze code. (Als je een sub-afdeling van een bestaande afdeling toevoegt is dit niet noodzakelijk).

Ook verder experimenteren met de rapporten die SampleApp aanmaakt kan interessant zijn. Kijk eens wat er verandert in het bestand met de sleutel 'file.ngwImport' als je het het hekje voor 'HomeDirectory' in `createparam.properties` verwijderd.

De demo-bestanden worden ook bij het ontwikkelen van CUG gebruikt. Mochten er reproduceerbare scenario's zijn die een ongewenst resultaat opleveren, dan wil ik dit graag weten (mail tverhagen@hsbos.nl [mailto:tverhagen@hsbos.nl?subject=CUG%201.0])

Java onderdelen verkennen

Om iets origineels te doen met CUG is gebruik van eigen java-classes noodzakelijk. CUG is ook een poging de problematiek van het account-beheer programmeerbaar te maken. Wil je hier mee verder dan is de java-code van de demo een goed beginpunt. Via de classes die hier gebruikt worden, kun je dan de api-documentatie benaderen die bij de source-distributie hoort. Begin bij Person en NDSUser, bekijk dan PTree en NDSMap en CugApp.

SampleApp maakt intensief gebruik van Configuration om de waarden uit de configuratie-bestanden te halen. Het kan lonend zijn hier ook eens naar te kijken. Voor het gebruik van jdbc-databronnen, kan de package jdbc een hulpmiddel zijn. Zie de broncode van de package cug.tables voor een werkwijze.

Een geschikte vingeroefening is het schrijven van een cug-rapport voor SampleApp. Maak een subclass van SimpleNDSMapRapport, definieer een constructor die een PrintWriter ontvangt en override report(NDSUser).

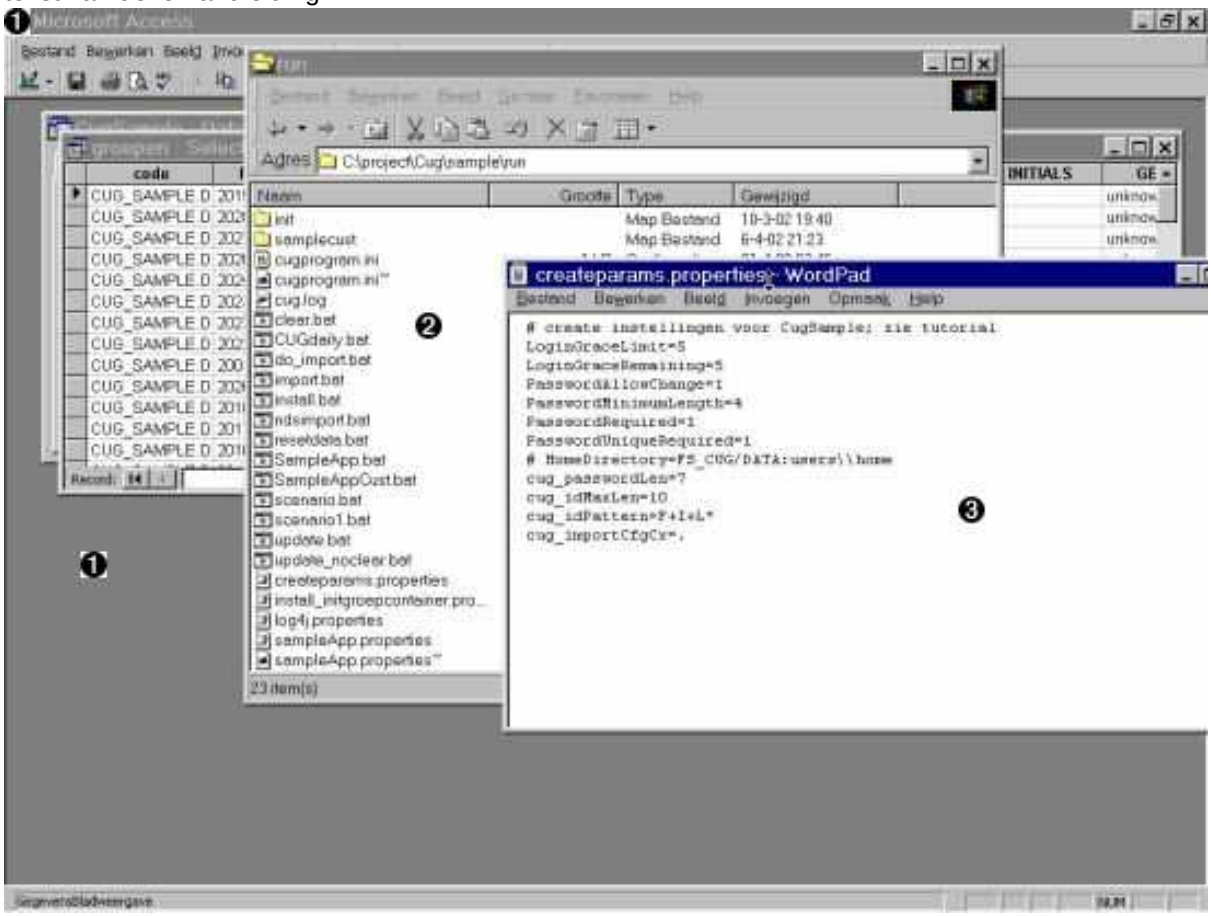
Als je verworven inzicht terugkoppelt naar de classes van de demo zul je erachter komen dat je 2 plugin-classes moet schrijven: een subclass van NDSContainer en een implementatie van CugOrg. Als je die hebt, kun je een rapportage toepassing maken die die gebruik maakt van een CugApp (vergeet niet je CugOrg te vermelden in `cugprogram.ini`).

Aanhangsel A. Typische demo-sessie

CUG heeft geen GUI, omdat CUG niet bedoeld is voor interactief gebruik. De configuratie van CUG is een neerslag van afspraken, niet het resultaat van een kwartiertje klikken en slepen door 1 persoon. In de praktijk is dit niet zo'n bezwaar, maar bij het doorlopen van de demo is dit wel lastig. Degene die met de demo werkt, is bezig met het uittesten van afspraken en speelt daarbij alle betrokken rollen.

Onderstaande afbeelding geeft een schermbeeld van een sessie waarin het CUG import/verwerken gedeelte wordt geconfigureerd. Als CUG een GUI had, hoefde je maar 1 venster te openen, nu zijn er dat dus 3 of meer. Het is dan het gemakkelijkst om Alt-Tab te gebruiken om van het ene naar het andere venster te gaan.

Naast de hier getoonde vensters is er meestal ook een browser-venster open met bijvoorbeeld de tekst van deze handleiding.



❶ Access met de CUG-databron geopend; bekijk hier het resultaat van de import en de vergelijking; voeg zelf queries toe.
❶ De map `run`; selecteer een batch-file en geef Enter om hem uit te voeren.
❷ Teksteditor met `createparam.properties`; wijzig een waarde en sla het bestand op.

Een sessie met rapporteren als onderwerp, heeft waarschijnlijk `sampleapp.properties` open in plaats van `createparam.properties` en naast `run` ook nog de map `out` waar de demo-rapporten in terecht komen.

Wellicht dat naar aanleiding van reacties op deze tekst duidelijk wordt wat een geschikt model is voor een GUI.

Aanhangsel B. CugReport kolomnamen

De class `cug.report.CugReport` wordt gebruikt om eenvoudige tabel-bestanden aan te maken met gegevens van een groep nds-users. Daarbij kunnen de kolommen tijdens *create time* ingesteld worden.

Voor CUG 1.0 zijn in ieder geval de volgende kolomnamen ter beschikking.

naam kolom	opmerking
context	naam van de container waar de user zich bevindt
dn	de uitgespelde naam van de user in de nds
email	
key	personid van nds-user
lastname	lastname van nds-usersw
name	naam van nds-object; user-name
password	
firstname	
formattedname	volledige naam
formattednameinitials	volledige naam met voorletters
formattedsortname	
formattedsortnameinitials	
initials	
nameinfix	tussenvoegsel
sortname	
fn	hetzelfde als formattedname
gender	sexe
personid	
userid	

CugReport zoekt hiervoor naar get...() methoden in bepaalde classes: via `cug.novell.NDSUser` wordt naar `util.Person` en diens `util.INames` gekeken. De 'get' wordt van de methodenaam afgehaald om kolomnamen te krijgen.

In bovenstaande tabel staan alleen namen voor get()-methoden die een String opleveren. Andere waarden worden met toString() geprint. Als een get() een *primitive type* oplevert, zal een fout optreden. Het veld 'status' is daarom niet geldig.