ABIS NIEUWSBRIEF VOOR DB2 PROFESSIONALS

JAARGANG 11 - NUMMER 2, JUNI 2019

OPEN CURSOR

Db2 for z/OS zit al een poosje stabiel op versie 12. Dit zal waarschijnlijk niet snel veranderen. Dat houdt ons uiteraard niet tegen om regelmatig wat Db2-nieuws voor u te bundelen.

Want natuurlijk blijft Db2 verder evolueren. En vooral ook de wereld omheen Db2, met name de data-wereld. In dat verband horen we steeds meer de term "data science" vallen. Een domein waarin Db2 (en andere databases) een belangrijke rol speelt. En die rol zal in de nabije toekomst alleen maar belangrijker worden.

Daarom dit themanummer, waarmee we u (misschien voor het eerst?) willen laten kennis maken met de twee belangrijkste programmeertalen in de data science arena.

Vergeet na het lezen van deze bijdrage niet, ons cursusaanbod te bekijken op pagina 15.

Alvast veel leesgenot,
Het ABIS Db2-team.

IN DIT NUMMER:

De twee bijdragen in dit nummer hebben als gemeenschappelijk thema: Db2-data ontsluiten voor "data science" gerelateerde statistische analyses:

- In "Db2 en Python" verneemt u hoe Db2 kan benaderd worden vanuit de populaire programmeertaal Python, en vooral wat het Python-ecosysteem als meerwaarde kan bieden voor de Db2-gebruiker.
- De tweede bijdrage, "Db2 en R", probeert u een beeld te geven van wat R is, en hoe u Db2-data in R kunt analyseren en visualiseren.

Gerelateerd: op de volgende IDUG-conference (oktober 2019 in Rotterdam) zal ABIS een presentatie geven over "Db2&Spark".

- En ten slotte, in "Dossier 12", een overzicht van de nieuwe "pagination"-syntax in Db2 12.

2

CLOSE CURSOR

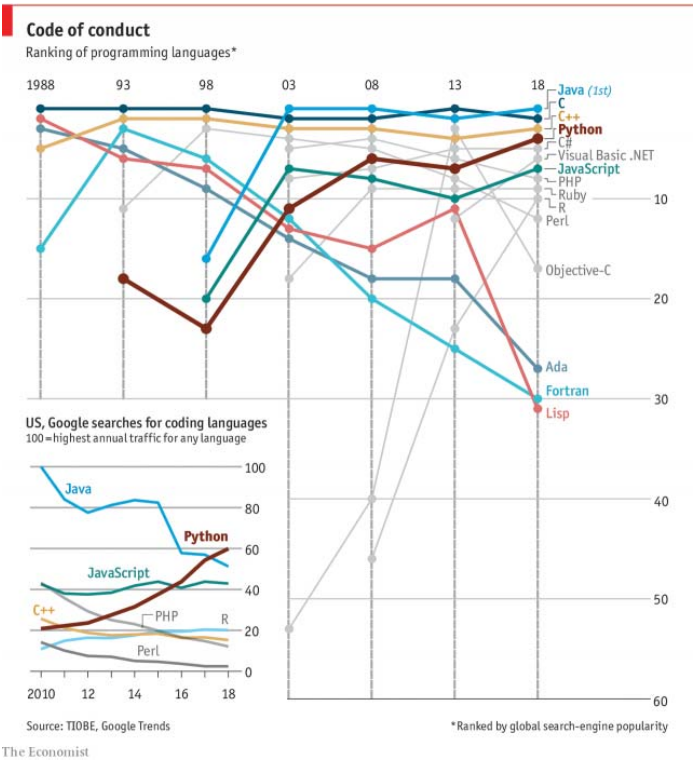
Ook in de volgende nummers van Exploring Db2 blijven we u op de hoogte houden van alle belangrijke ontwikkelingen i.v.m. Db2, en ook (in de ruimere zin) van data(bases) en data analytics.

Over welke onderwerpen wilt u meer lezen? Meld het ons: *training@abis.be*

Db2 en Python *Arnout Veugelen (ABIS)*

In tijden van DevOps en cross-functional tribes, blijven we als IT-professional steeds minder op het eilandje van onze expertise. Van programmeurs en data-analisten verwachten we dat ze met een database overweg kunnen, en klassieke infra-profielen worden aangevuld met programmeerskills.

Als het op de keuze van een programmeeromgeving aankomt, blijkt Python erg populair: alle indicatoren wijzen erop dat het gebruik van Python stevig in de lift zit, en scholen en universiteiten kiezen massaal voor Python als leerplatform.



Bron: The Economist, 26 juli 2018, Python is Becoming the most popular Programming Language: https://www.economist.com/graphic-detail/2018/07/26/python-is-becoming-the-worlds-most-popular-coding-language

---

Python is bijna dertig jaar oud, en is in die tijd gegroeid van een niche-scripting-taal voor OS-operaties naar de eerste keus van heel wat programmeurs en niet-programmeurs voor diverse taken. Twee belangrijke katalysatoren in die groei: de opgang van internetbedrijven sinds het einde van de jaren 1990 en de huidige interesse voor data science en aanverwante.

Hoge productiviteit dankzij elegante syntax

Men oppert wel eens dat Python relatief *traag* is. Dat klopt in zekere zin: als je naar uitvoeringstijd van programma's gaat kijken, zijn de cijfers voor Python niet indrukwekkend. Dat heeft te maken met het feit dat het een geïnterpreteerde taal is, en dat het vertalen van broncode in Python naar machinecode een vrij arbeidsintensieve klus is. Een bewuste keuze: Python is ontworpen om dicht bij de programmeur te staan, eerder dan dicht bij de machine: traag in uitvoering, maar snel in ontwikkeling. Een denkpatroon vertalen naar Python is over het algemeen eenvoudig, de resulterende programmacode is leesbaar en compact. Vergelijk bijvoorbeeld het volgende stukje code in Python met een equivalent programma in C:

Creëer een collectie van gehele getallen. Print vervolgens de getallen uit de collectie die deelbaar zijn door drie. Print 'spam!' voor de overige getallen.

Python:

```python
values = [1, 17, 313, 28, 99, 12, 4238]

for item in values:
    if item % 3 == 0:
        print(item)
    else:
        print('spam!')
```

C:

```c
#include <stdio.h>

int main() {
  int values[] = {0, 1, 17, 313, 28, 99, 12, 4238, EOF};
  for (int *item = values; *item != EOF; ++item)
    if (*item % 3 == 0)
      printf("%d\n", *item);
    else
      printf("spam!\n");
  return 0;
}
```

Naast deze comfortabele basissyntax en krachtige built-in objecten, beschikt Python over een ruime *standard library*: verzamelingen van standaardobjecten en procedures voor allerlei taken, die je zonder moeite in een programma kan inpluggen. Een rijk ecosysteem aan *third party libraries* voorziet bovendien oplossingen voor specifieke toepassingen.

Op deze manier kan een programmeur erg snel en efficiënt werken: de uitvoeringstijd van het programma is misschien niet spectaculair snel, de ontwikkeltijd voor de programmeur (en bijgevolg de productiviteit van het bedrijf) is dat wel. Zo is Python een zeer interessante keuze voor allerlei (internet-)technologiebedrijven, waarvoor snelle innovatie onontbeerlijk is. Dat verklaart de groei in populariteit rond het jaar 2000.

Data Science

Ook de huidige opgang van Python is in de eerste plaats te danken aan programmeercomfort. Dankzij de heldere syntax is het ook voor een buitenstaander vrij makkelijk om met Python aan de slag te gaan, en zo wint de taal aan populariteit onder niet-IT'ers.

Wetenschappelijk onderzoek bijvoorbeeld gaat tegenwoordig bijna altijd gepaard met het analyseren van grote hoeveelheden data, en daarvoor dienen computerprogramma's geschreven te worden; vaak in Python. Nieuwe planeten ontdekken? Python. De eerste foto van een zwart gat? Python.

Ook voor business-analisten, economen, marketeers etc. is data-analyse onontbeerlijk geworden. En hoewel Python qua opzet meer *general purpose* is dan gespecialiseerde talen zoals bijvoorbeeld SAS of R, is het ook voor statistische analyses, Machine Learning en Artificial Intelligence een zeer interessante keuze, dankzij libraries zoals Numpy (krachtige multi-dimensionele arrays), Pandas (spreadsheet-achtige DataFrames), Matplotlib (visualisatie) en scikit-learn (Machine Learning). Modieuze tools voor Deep Learning (zoals Tensorflow en Keras), en Big Data-platformen zoals Spark zijn in meer of mindere mate op Python gebaseerd, of bieden Python op zijn minst aan als gebruikersinterface.

Python en Db2

Het is over het algemeen een koud kunstje om een Python-programma te koppelen aan een databron. Er zijn interfaces beschikbaar voor zowat alle populaire databases, en Db2 vormt daarop geen uitzondering: er zijn zelfs een aantal verschillende mogelijkheden. Zo is er *PyODBC* dat, u raadt het al, een Python-programma koppelt aan een ODBC-databron naar keuze, wat met name binnen een Windows-omgeving een interessante optie is. Mogelijk nog interessanter, want ook vlot bruikbaar op andere besturingssystemen, is *ibm_db*, ontwikkeld door IBM.

Hoe ga je ermee aan de slag?

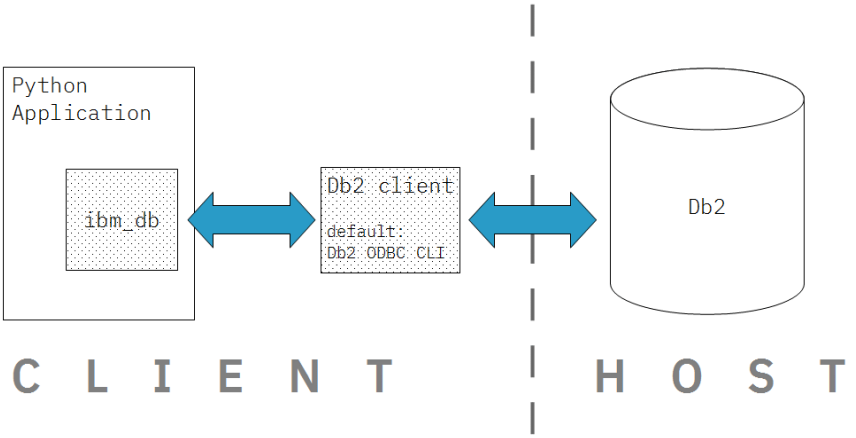
- Installeer Python. Standaard-implementaties van Python (CPython) voor zowat elk besturingssysteem vind je terug via www.python.org.
  In de meeste Linux-distributies is CPython reeds voorgeïnstalleerd. Er bestaan ook andere Python-implementaties, zoals Jython (compileert naar Java-machinecode) en PyPy (Just-In-Time-compilatie).
  Ook het overwegen waard: een repackaged CPython-distributie,

zoals bijvoorbeeld Anaconda, met een heleboel reeds geïnstalleerde libraries voor Data Science: *www.anaconda.com*.

- Installeer `ibm_db`. IBM ontwikkelde `ibm_db` als Python interface voor Db2 (LUW, z/OS en i5) en Informix. Via de Python package manager *pip* download én installeer je `ibm_db` simpelweg via de command line:

```
pip install ibm_db
```

- Als standaard Db2-client wordt de IBM ODBC and CLI Driver (cli-driver) automatisch meegeïnstalleerd, maar desgewenst kan je `ibm_db` ook koppelen aan een andere Db2-client.
(Voor meer details en aanwijzingen i.v.m. het koppelen van uw Db2-license: zie *github.com/ibmdb/python-ibmdb#-installation*)



`Ibm_db` ondersteunt in feite twee verschillende interfaces: *ibm_db* en *ibm_db_dbi*. De eerste is gemaakt naar de eigen inzichten van IBM, de tweede volgt de *Python DB-API 2.0*-specificatie.

Deze specificatie, ook bekend als *Python Enhancement Proposal (PEP) 249*, is bedoeld om een uniforme interface tussen Python en verschillende database-platformen mogelijk te maken. PEP 249 omschrijft standaarden voor de objecten en de functionaliteiten die een Python-database-interface zou moeten bevatten. Voor zowat elke courant voorkomende database-server bestaat er intussen zo'n compatibele module, en dat is zeer interessant voor de ontwikkelaar: elke database laat zich zo op dezelfde manier aanspreken, en je hoeft een programma nauwelijks aan te passen bij

een migratie naar een andere back-end. De bepalingen van PEP 249 (en dus ook van `ibm_db_dbi`) vind je terug op *www.python.org/dev/peps/pep-0249*.

Eens geïnstalleerd, kan je `ibm_db` simpelweg in je Python-programma importeren met een `import` statement en zet je een connectie op met `connect( parameters... )`. Een ibm_db-cursor-object komt min of meer overeen met een Db2-cursor. Je creëert het met de gelijknamige methode. De cursor kan vervolgens SQL-statements uitvoeren.

Ter illustratie een uittreksel uit een interactieve Python-sessie:

```
#1
>>> import ibm_db_dbi

#2
>>> conn_str = 'database=dev;hostname=test.abis.be;port=50000;protocol=tcpip;'
>>> conn = ibm_db_dbi.connect(conn_str, user='arnout', password='******')

#3
>>> cur = conn.cursor()

#4
>>> query = """select id, name from tbaccad.tutpersons"""
>>> cur.execute(query)
True
>>> cur.description
[['ID', DBAPITypeObject({'SMALLINT', 'INTEGER', 'INT'}), 6, 6, 5, 0, False],
['NAME', DBAPITypeObject({'CHARACTER', 'CHARACTER VARYING', 'STRING', 'CHAR',
'VARCHAR', 'CHAR VARYING'}), 45, 45, 45, 0, True]]

>>> results = cur.fetchall()

#5
>>> for row in results:
        print(row[0], row[1], sep = ': ')

1: Marckx
2: Zureroom
3: Petanquil

#6
>>> conn.close()
True
```

(#1) importeer ibm_db_dbi
(#2) Zet een Db2-connectie op
(#3) Maak een cursor-object
(#4) Laat het cursor-object een SQL-query uitvoeren, sla de output op in een Python-object. Het description-attribuut van de cursor bevat de metadata (kolomnaam, datatype...)
(#5) Loop door de resultaatsset
(#6) Sluit de database-connectie

Merk op dat een interactie met behulp van `PyODBC` in plaats van `ibm_db_dbi`, of zelfs met een andere Python-module naar een totaal andere database-server (Oracle, MySQL, SQL Server, MongoDB ...) er exact hetzelfde zou uit zien, op de eerste twee stappen na (de import en de connectiestring).

Het bovenstaande voorbeeld illustreert de klassieke aanpak: een query op de database wordt uitgevoerd en de resultaten worden zo goed als mogelijk vertaald naar standaard Python-objecten: elk record in de resultaatstabel wordt een *tuple* (een geordende collectie), en elke value in zo'n record wordt omgezet naar het meest geschikte objecttype: integers, strings etc.

In een handomdraai zet je één en ander om naar complexere, rijkere datatypes, zoals b.v. *dictionaries*, met key-value-paren:

```
#1
>>> cur.execute("""select first_name, name, func from tbaccad.tutpersons""")
True
>>> column_names = [column[0] for column in cur.description]
>>> column_names
['NAME', 'FIRST_NAME', 'FUNC']

#2
>>> results = cur.fetchall()
>>> persons = []
>>> for row in results:
        person = dict(zip(column_names, row))
        persons.append(person)

>>> persons
[{'NAME':'Marckx', 'FIRST_NAME':'Rudy', 'FUNC': 'General manager'},
 {'NAME':'Zureroom', 'FIRST_NAME':'Jaap', 'FUNC': 'Developer'},
 {'NAME':'Petanquil', 'FIRST_NAME':'Jean', 'FUNC': 'Architect'}]

#3
>>> persons[0]['FIRST_NAME']
'Rudy'
```

(#1) Met een zogenaamde "list comprehension" verzamelen we de kolomnamen uit de cursor-description.
(#2) We lopen door de resultaten van de query. Voor elke record maken we een dictionary-object. De dictionaries verzamelen we in een list-object genaamd persons.
(#3) Een dictionary bestaat uit key-value-paren: met behulp van de sleutel 'FIRST_NAME' kunnen we de bijhorende value opvragen.

Alchemie en pandas

De voorgaande snippets leggen de objectgeoriënteerde natuur van Python bloot: objecten worden gecreëerd en vervolgens via objectmethodes aangesproken. Python is inderdaad een zuivere OO-taal, maar laat desondanks ook een sequentiële, imperatieve programmeerstijl toe: je kan perfect programma's schrijven zonder zelf ooit een *class* te moeten definiëren. Voor wie dat wel wil doen, en een rechtstreekse, duurzame (persistent) link wil opzetten tussen

enerzijds class instances in een programma en anderzijds table records in een database, is *Object Relational Mapping* (ORM) de oplossing. Dat wordt mogelijk door `ibm_db` uit te breiden met *SQLAlchemy*.

Ook de *pandas* bibliotheek is het vermelden waard: die levert namelijk *DataFrames*, tabellen met rij- en kolomindexen, en krachtige functies voor filteren, samenvoegen en samenvatten. Allemaal concepten die zich uitstekend laten vertalen naar de context van een Db2-database en dus erg interessant als u bijvoorbeeld data uit de database wil combineren met andere databronnen. Pandas DataFrames zijn trouwens zeer vergelijkbaar met data frames in R, waarover u meer leest in het volgende artikel.

Verder ontdekken?

Alles bij elkaar vormt Python dus een erg interessante uitbreiding op uw database-kennis. Is uw nieuwsgierigheid geprikkeld? Kom dan bij ons verder kennis-maken met Python, tijdens een van onze Python-opleidingen, waaronder:

- *Python: basiscursus*
- *Python voor data-analyse*
- *Intermediate Python*

Voor wie zijn of haar allereerste programmeerstappen wil zetten, vermelden we ook nog graag onze training '*Programmeren: basiscursus*', waarin we de belangrijkste programmeerconcepten in de praktijk brengen met behulp van Python.

Db2 en R

Peter Vanroose (ABIS)

Zoals we allemaal weten is Db2 —zowel op z/OS als op LUW— het ideale platform voor het gestructureerd bijhouden en efficiënt ondervragen van omvangrijke datavolumes, zowel in een "data warehouse"-context als transactioneel.

SQL (en dus Db2) laat bovendien op een eenvoudige manier toe die data te *aggregeren* en dus samen te vatten voor statistische doeleinden. Maar wat doen we best wanneer we net een stapje verder willen gaan, b.v. die samenvattingen visualiseren, of statistisch significante analyses of voorspellingen maken?

Dan is een statistisch pakket nodig, maar moeten we dan alle Db2-data overpompen, en hoe kan dat op een efficiënte manier?

Wat is R?

R is een populair statistisch software-pakket, vrij en open-source (*https://www.r-project.org/*), met een rijke set ingebouwde data-analyse- en visualisatie-mogelijkheden. Mijn persoonlijke eerste kennismaking met R dateert nog uit de jaren 90. Daarna werd deze software wat "vergeten" (niet alleen door mij!), maar sinds enkele jaren is R weer populair dankzij de beschikbaarheid van een uitgebreide set aan zgn. *packages* (zeg maar: extensie-bibliotheken) o.a. voor toepassingen in Big Data en Data Science.

Data Science, Big Data, en Db2

In *nummer 2 van de 9de jaargang van Exploring Db2* hebben we u al bijgepraat over de rol van Db2 in het Big Data-landschap. Daarin werd vooral gefocust op enerzijds de mogelijkheden binnen Db2, en anderzijds de rol die infrastructuur zoals Hadoop en NoSQL-databases kunnen spelen voor het toegankelijk maken van een verscheidenheid aan databronnen voor data-analyse.

Ondertussen is het duidelijk geworden dat het in het groeiende en veelzijdige aanbod aan tools voor data science vooral belangrijk is, deze software-pakketten, en ook de verschillende databronnen, gemakkelijk *aan elkaar te kunnen koppelen*.

Dit geldt dus ook voor Db2 enerzijds, met z'n krachtige relationele query-optimizer, en de (meestal niet-relationele) volumineuze data buiten Db2 anderzijds: hoe koppelen we die twee aan elkaar, en hoe presenteren we de resultaten van een statistische data-analyse overzichtelijk aan de eindgebruiker?

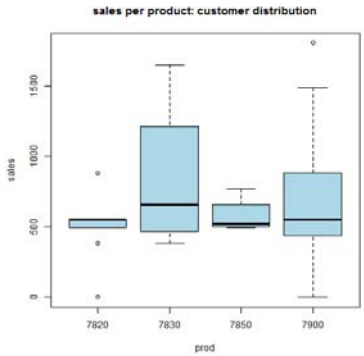
R als visualisatie-tool

Zonder in detail te gaan, kunnen we zeggen dat R (dat in de eerste plaats statistische software is) vooral ook een gemakkelijk te gebruiken visualisatie-tool is. Output komt by default op het scherm, maar kan ook als PDF of als image (GIF, PNG, ...) opgeslagen worden.

Enerzijds is er de generieke `plot()`-functie, die de meest aangewezen grafiek maakt voor z'n data-argument. Voor b.v. een tabel van twee numerieke kolommen (zoals lengte en gewicht) wordt een spreidingsdiagram (scatter plot) getekend; voor een tabel met datums en getallen wordt een trendlijn getekend; voor een tabel met een discrete eerste kolom (een zgn. "factor") worden meerdere boxplots getekend (zie voorbeeld hieronder); ...

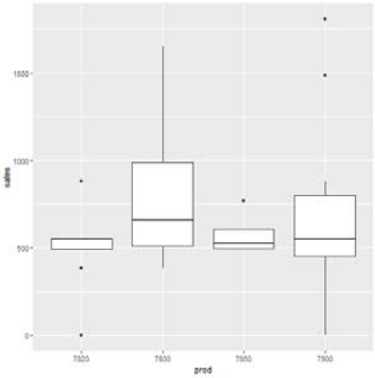
Een aantal (eenvoudige en minder eenvoudige) opties kunnen met `plot()` meegegeven worden om kleur, legende, lijndikte, datapunten, ... op te geven. Een simpel voorbeeld:

```
pdf('sales.pdf') # output zal naar het bestand "sales.pdf" geschreven worden
# alternatief is b.v. png('sales.png')
plot(sales.data, col='lightblue', main='sales per product: customer distr')
# (tabel met twee kolommen: product + omzet per klant)
dev.off()  # "sluit" de output naar het PDF-"device"
```



Een redelijk recent grafisch R-package is ondertussen populair geworden en is *de facto* `plot()` aan het verdringen: `ggplot2()`. Z'n grafieken ogen moderner, de grafische mogelijkheden zijn nog uitgebreider (en voor "standaard" grafieken toch gebruiksvriendelijk)

```
install.packages('ggplot2')
ggplot2::ggplot(sales.data, aes(x=prod, y=sales)) + geom_boxplot()
```



R als datarepresentatie-tool

Een "tabel" in R heet een **data frame**; dit is technisch een list ("tuple") van (homogene) vectoren van dezelfde lengte, de kolommen dus. Ongeveer alle "high-level" data-interfaces van R werken met data frames as datatype. Net zoals bij een relationele tabel heeft elke vector van een data frame (elke kolom dus) een naam. Een data frame is weliswaar geordend: elke tuple (rij dus) heeft impliciet een rijnummer.

R is 100% object-georiënteerd, dus een data frame object hoeft z'n data niet in het werkgeheugen te laden (als tuple van vectoren), zolang het zich maar als dusdanig gedraagt. R gebruikt dus (waar mogelijk) de zgn. *lazy evaluation*-strategie. De gebruiker kan zelf een functie definiëren die via data frames communiceert, maar intern zo efficiënt mogelijk werkt.

Een prototype functie in dit verband is `read.csv`, die een CSV-bestand inleest en een object van type data.frame teruggeeft, b.v.:

```
tbl = read.csv('sales.csv', sep=';', header=TRUE)
```

of syntactisch equivalent (en typischer voor R):

```
tbl <- read.csv('sales.csv', sep=';', header=TRUE)
```

Dit object kan dan aan statistische functies meegegeven worden, en natuurlijk ook aan `ggplot()`, die er de informatie uit extraheren die ze nodig hebben, opnieuw via ingebouwde (low-level) functies.

Bruggen bouwen tussen R en Db2

IBM ondersteunt sinds enkele jaren het gebruik van R, welswaar vooral in een cloud-context, i.h.b. dashDB. De meeste documentatie (bv. *support.labs.cognitiveclass.ai/knowledgebase/articles/916482*)

is dan ook vooral bedoeld om enkele IBM-producten te promoten, o.a. BigInsights. Maar met een kleine inspanning en wat kennis van R kan het ook met alleen maar een lokaal geïnstalleerde R, en uiteraard een connectie naar Db2 (voor z/OS of voor LUW).

Dankzij de object-georiënteerde structuur van R is het niet zo moeilijk om de low-level details van de communicatie tussen R en Db2 in te pakken in high-level objecten zoals data frames.

"Low-level" betekent in dit geval: standaard client-server protocols voor (relationele) database communicatie, zoals ODBC of JDBC.

RODBC en RJDBC

Wie op Windows werkt, zal ODBC als meest "natuurlijke" keuze zien voor het opzetten van een database-connectie, b.v. naar Db2. Dit kan dan een lokale Db2 (LUW) zijn, of een connectie naar een (z/OS) host, al dan niet via een Db2 Connect Gateway.

Op Linux of Unix is een JDBC-setup gebruikelijker, of beter gezegd: is het opzetten van ODBC niet zo evident.

In beide gevallen moet Db2 als "databron" gedefinieerd worden. Zie details in b.v. *developer.ibm.com/articles/dm-1402db2andr/*.

Gelukkig ondersteunt R beide, door middel van twee packages: **RODBC** en **RJDBC** respectievelijk. Installeren (eenmalig) van een R-package is zeer eenvoudig, zoals hierboven al duidelijk werd (voor ggplot2). RODBC installeren gebeurt dus met één R-instructie:

```
install.packages('RODBC')
```

Na de installatie zijn de RODBC-instructies (of functies) beschikbaar door de `RODBC::` prefix te gebruiken, of (wat gebruikelijker is) door alle RODBC-functies te "laden" in de "default namespace" met:

```
library(RODBC)
```

Er komen een 30-tal functies mee met het RODBC-package; hun namen beginnen met "odbc". Voor de officiële documentatie, zie *https://cran.r-project.org/web/packages/RODBC/vignettes/RODBC.pdf*

Enkele voorbeelden:

```
dbh <- RODBC::odbcConnect('DB2T', uid='Peter', pwd='******')
```

Deze "database handle" moet verder gebruikt worden voor ongeveer alle andere ODBC-functies; dit laat o.a. toe, meerdere database-connecties in parallel te openen, b.v. om data van verschillende database servers te combineren.

De "`RODBC::`" prefix kan weggelaten worden dankzij de `library()` instructie van daarnet. Dat doen we verderop dan ook.

Om een SQL-query naar Db2 te sturen, b.v. om de eerder genoemde verkoopsstatistieken op te halen als data frame, schrijven we:

```
querytext <- 'SELECT s_cid AS prod, SUM(epay) AS sales
                FROM tutsessions JOIN tutenrolments ON e_sno=sno
              GROUP BY s_cid, e_cono'
odbcQuery(dbh, querytext)
sales.data <- as.data.frame(odbcFetchRows(dbh)$data)
names(sales.data) <- c('prod','sales')
sales.data$sales <- as.numeric(as.character(sales.data$sales))
```

Merk op dat een "." een geldig teken is in de naam van een R-variabele, en dat "$" nodig is als separator vóór de kolomnaam.

RJDBC werkt gelijkaardig; het voorbeeld van daarnet zou er nu als volgt uit zien:

```
install.packages('RJDBC') ; library(RJDBC)
db2conf <- JDBC('com.ibm.db2.jcc.DB2Driver','/path/to/db2jcc4.jar')
dbh <- dbConnect(db2conf,'jdbc:db2://10.24.1.1:50000/DB2T:user=Peter;...')
sth <- dbSendQuery(dbh, querytext)  # zelfde tekstvariabele van hierboven
sales.data <- fetch(sth)
dbDisconnect(dbh)
```

`sales.data` is nu automatisch een data frame met kolomnamen `prod` en `sales`, klaar dus voor `plot()` of `ggplot()`, nadat we de tweede kolom numeriek gemaakt hebben (zoals bij RODBC). Documentatie voor RJDBC: *cran.r-project.org/web/packages/RJDBC/RJDBC.pdf*.
Bemerk dat RJDBC het generieke database interface framework **DBI** gebruikt; zie *cran.r-project.org/web/packages/DBI/DBI.pdf*.
Daar is de documentatie te vinden voor b.v. `dbSendQuery()`; anderzijds is b.v. `fetch()` een specifieke RJDBC-functie.

Encapsulatie

Bemerk dat we (jammer genoeg) de output van `odbcFetchRows` expliciet naar een data frame moesten converteren, de kolommen expliciet een naam moetsen geven, en daarna expliciet de tweede kolom numeriek moesten maken.

```
dataframe_from_db2 <-
  function(stmt, col.names) {
    dbh <- RODBC::odbcConnect('DB2T', uid='Peter', pwd='******')
    RODBC::odbcQuery(dbh, stmt)
    df <- as.data.frame(RODBC::odbcFetchRows(dbh)$data)
    names(df) <- col.names
    df[[2]] <- as.numeric(as.character(df[[2]]))
    return (df)
  }
```

die we daarna (eventueel meermaals) kunnen oproepen:

```
sales.data <- dataframe_from_db2(querytext, c('prod','sales'))
```

De boxplot per product (cursus in dit geval) krijgen we ten slotte met de reeds getoonde `plot()` of `ggplot()` instructies.

De volgende stap

Uit bovenstaand voorbeeld kunnen we alvast leren dat het (meestal) niet nodig is om volledige tabellen “over te pompen” van Db2 naar de visualisatie- of analyse-omgeving. In het voorbeeld werd de data reeds gedeeltelijk geaggregeerd, namelijk op de combinatie van klant & product, om dan in R een beeld te krijgen van de verdeling (over klanten heen) van elk product. Deze “rollup” van de afzonderlijke verkoopcijfers levert gemakkelijk winst in datavolume op van enkele gigabytes naar enkele megabytes.

Hadden we verder kunnen gaan, en het netwerk-trafiek van Db2 naar R nog meer reduceren, door een bijkomende “rollup” van de klantendetails? Kort antwoord: neen, want R heeft deze details nodig om de statistische verdeling te berekenen, nodig voor het tekenen van b.v. de boxplots hierboven.

Een iets gedetailleerdere analyse zou echter leren dat er voor het tekenen van exact dezelfde grafieken maar een beperkt aantal getallen (per product) nodig zijn: voor een boxplot zijn dat de mediaan, de twee kwartielen, en eventueel enkele outliers.

IBM heeft alvast een (kleine) stap gezet in deze richting van minder dataverkeer tussen Db2 en R: zie hun R-package genaamd *ibmdbR*, dat ze vooral in de context van *dashDB* promoten. Hierin worden berekeningen (van b.v. de mediaan) gedelegeerd naar Db2.

Meer weten?

Nieuwsgierig geworden naar wat R nog te bieden heeft? Wilt u zelf aan de slag met R? Een aanrader: volg onze introductie-cursus *“R voor data-analyse”*.

DOSSIER 12

Nieuwe "pagination"-syntax

De term "pagination" wordt gebruikt in de context van potentieel zeer lange result sets (cursor) die we vanuit een applicatie in "beetjes" willen aanbieden aan de gebruiker, zeg maar: scherm per scherm. Denk b.v. aan een Google Search, waarbij het eerste scherm slechts een 10-tal resultaten toont, en waarbij je onderaan de pagina de kans krijgt naar de volgende (of zelfs dadelijk naar b.v. de 7de) pagina te springen.

Het ophalen van de eerste pagina is eenvoudig: voeg een `FETCH FIRST 10 ROWS ONLY` toe aan de query die anders alle data zou opvragen.

Het concept van "volgende" of "10de" pagina is pas mogelijk wanneer er voor een bepaalde sortering van de data gekozen is. Want dan krijgen we "alle volgende data" (en dus ook "de volgende pagina") door een extra WHERE-conditie, nl.: alle rijen waarvoor *de waarde van de sorteersleutel strikt groter is dan de laatst geziene*.

Voor een sleutel van één kolom is dat eenvoudig: stel dat dit de kolom k is, en we hebben net als laatste rij van de pagina de waarde k=510 getoond. Dan wordt de data voor de volgende pagina gegeven door iets als:

```
SELECT <kolommen> FROM <tabel>
 WHERE <conditie>   AND k > 60
ORDER BY k     FETCH FIRST 10 ROWS ONLY
```

(Uiteraard zullen de "60" en "10" typisch door host-variabelen vervangen worden.) Wanneer kolom k geïndexeerd is, wordt dit trouwens een (zeer) efficiënte query.

Het klassieke "pagination"-probleem ontstaat pas wanneer die sleutel uit twee of meer kolommen bestaat. Zelfs indien er een index bestaat op deze kolomcombinatie. Hoe kunnen we ervoor zorgen dat we deze index telkens verder doorlopen?

De "simplistische" implementatie doet dit in elk geval niet:

```
SELECT <kolommen> FROM <tabel>
 WHERE <conditie>     AND   k1 || k2   >   :laatste_k1 || :laatste_k2
ORDER BY k1, k2   FETCH FIRST 10 ROWS ONLY
```

want die "`k1||k2`" concatenatie is uiteraard niet indexeerbaar! We krijgen dus een massieve materialisatie van de hele tabel, gevolgd door een massieve sortering...

De "klassieke" aanpak is de volgende: vervang dat extra predikaat door:

```
   AND (     k1 = :laatste_k1  AND  k2 > :laatste_k2
         OR  k1 > :laatste_k1)
```

De nieuwe v12-syntax is een stuk eenvoudiger, met minder gevaar voor vergissingen:

```
   AND (k1,k2) > (:laatste_k1, :laatste_k2)
```

Conceptueel weer de concatenatie, maar nu met 100% zekerheid van indexeerbaarheid!

Ook kunnen we nu eenvoudig dadelijk naar b.v. de 7de pagina te springen:

```
  ORDER BY k1, k2  OFFSET 60 ROWS   FETCH FIRST 10 ROWS ONLY
```

Peter Vanroose

CURSUSPLANNING, JUNI - DEC 2019

SQL & relationele databases: basiskennis	850 EUR	5.09(L), 10.10(W), 7.11(L), 21.11(W)
Db2 for z/OS basiscursus	1425 EUR	16.09(L), 27.11(W)
Db2 for LUW basiscursus	1425 EUR	op aanvraag
SQL workshop	900 EUR	24.06(L), 7.10(L), 30.10(W), 5.12(L)
SQL voor gevorderden	500 EUR	18.06(W), 6.11(L), 20.12(W)
SQL voor BI en Data Science	950 EUR	26.09(W), 12.12(L)
SQL PL database programmeren	1000 EUR	15.05(L)
Db2 triggers, stored procedures, en User-Defined Functions	500 EUR	28.10(W)
Db2 for z/OS: programmeren voor gevorderden	1000 EUR	06.06(L), 25.11(L)
Db2 for z/OS: SQL performance	1500 EUR	24.06(L), 09.12(L)
XML in Db2	500 EUR	op aanvraag
Db2 for z/OS: database administratie	2100 EUR	14.10(L), 16.12(W)
Db2 for z/OS: installation & migration	1080 EUR	op aanvraag
Db2 for z/OS: data recovery	1020 EUR	op aanvraag
Db2 for z/OS: monitoring & tuning systems' performance	1050 EUR	01.07, 19.09 (virtual classroom)
Db2 for LUW DBA - Kernvaardigheden	2000 EUR	17.12(W)
Db2 11 for z/OS: changes & new features	525 EUR	op aanvraag
Db2 12 for z/OS: changes & new features	525 EUR	29.10(W), 12.11(L)
Moderne data warehousing & BI	500 EUR	16.09(W), 18.10(L)
Big Data architectuur & infrastructuur	500 EUR	19.06(W), 27.06(L)
R voor data-analyse	1425 EUR	09.10(L), 04.11(W)
Python voor data-analyse	950 EUR	03.10(W)
Big data in de praktijk met Spark	1000 EUR	28.05(L)
Big Data in de praktijk: text analytics	475 EUR	18.10(L)
Regular expressions	275 EUR	28.06(L), 30.10(W), 04.12(L) (avondsessies)

Plaats: (L) = Leuven, (W) = Woerden;
alle cursussen ook beschikbaar op aanvraag en/of op uw lokatie;
Voor details en andere cursussen, zie `https://abis.be/html/nlall.html`
Pour détails et d'autres cours, voir `https://abis.be/html/frall.html`
For details and other courses, see `https://abis.be/html/enall.html`