Fachbereich 07 Informatik/Mathematik



Praktikum Datenbanksysteme II

Wintersemester 2018/19

Prof. Dr. Martin Staudt

Übung 3

Wimmer, Anja

IF8

Gabl, Daniel

IF6

21.01.2019

# Inhaltsverzeichnis

[Inhaltsverzeichnis](#_Inhaltsverzeichnis) II

[Aufgaben](#_Aufgaben) 1

[Aufgabe 1](#_Aufgabe_1) 1

[Aufgabe 2](#_Aufgabe_2) 1

[Aufgabe 3](#_Aufgabe_3) 2

[Quellcode](#_Quellcode) 8

# Aufgaben

## Aufgabe 1

-

## Aufgabe 2

Bei der Implementierung der Min-Max-Skallierung haben wir zuerst eine Tabelle von Personen aufgestellt und mit Test-Werten gefüttert. Als zu skallierenden Wert haben wir das Alter der Person genommen (auch wenn dies nicht wirklich viel Sinn ergibt).

Die Skallierungsfunktion haben wir auch als eigene Funktion definiert, die eben die fünf Parameter (aktueller Wert, altes Minimum, altes Maximum, neues Minimum, neues Maximum) übergeben bekommt und aus der gegebenen Formel das neue Minum berechnet.

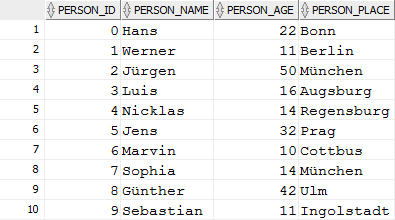
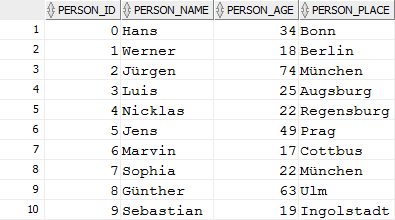
Formel d. Min-Max-Skallierung:

Beispiel:

Wir wollen eine Skala, die ursprünglich von 17 bis 74 ging, neuskallieren. Die neue Skala soll nun von 10 bis 50 gehen, als aktuellen Wert bekommen wir 34.

Dazu haben wir eine Prodezur geschrieben, die nur das neue Minimum und das neue Maximum übergeben bekommt, das alte Minimum und das alte Maximum wird mittels den Aggregatfunktionen MIN und MAX aus der Tabelle berechnet. Danach wird auf jeden Datensatz der Tabelle die Min-Max-Skallierung mit den übergebenen und berechneten Daten angewandt, wobei der aktuelle Wert dem Alter im aktuellen Datensatz entspricht. Nach der Berechnung des neuen Werts wird der alte Wert mit dem neuen Wert überschrieben.

Hier Screendumps unserer Tablle vor der Skallierung und nach einer 10,50-Skallierung:



Als Anmerkung: Eine sinnvolle Anwendung einer Min-Max-Skallierung ist, wenn man bspw. den „Fortschritt“ des aktuellen Jahres in Prozent berechnen möchte (26.12 = 98.36%).

Der Unix-Timestamp vom 01.01, Mitternacht des aktuellen Jahres = altes Minimum,

Der Unix-Timestamp vom 01.01, Mitternacht des nächsten Jahres = altes Maximum,

0 = neue Minimum, 100 = neue Maximum & der aktuelle Unix-Timestamp = aktueller Wert.

## Aufgabe 3

Aufgabe ist es, unser Data Warehouse mit Daten aus anderen Quellen zu füllen, dabei müssen auch Aktualisierungen korrekt gehandhabt werden.

Zuerst haben wir die vorgegeben Tabellen (Quellen und Zieltabelle) erzeugt und die Quelltabellen mit Beispieldatensätzen gefüttert. Dabei ist uns aufgefallen, dass wir einige Attribute der Zieltabelle nicht in jedem Fall oder nicht in der gewünschten Form haben und wir diese erst noch erzeugen oder uns zurechtlegen müssen, Beispiele dafür:

* Arbeiter haben kein Geschlechter-Attribut
* Angestellte haben ein Attribut, in dem Vor- und Nachname zusammenhängen
* Angestellte und Arbeiter haben nur ein Geburtsdatum, kein Alter
* Angestellte und Arbeiter haben nur Stunden / Monatslohn, kein Jahreseinkommen

Um diese Probleme zu beheben haben wir Hilfstabellen angelegt und Funktionen / Prozeduren geschrieben, die bspw. aus einem Geburtsdatum ein Alter errechnet.

Um das Geschlecht eines Arbeiters bestimmen zu können, haben wie eine Tabelle angelegt, die einen Vornamen auf ein Geschlecht mappt. Wenn wir einen Angestellten in unser Data Warehouse einfügen, nehmen wir seinen Vornamen und sein Geschlecht und speichern es in dieser Hilfstabelle. Fügen wir nun einen Arbeiter hinzu und der Vorname steht in unserer Hilfstabelle, so können wir daraus schließen, welches Geschlecht dieser hat. Wichtig hierbei ist es, dass der Name genau einmal in der Hilfstabelle vorkommt, denn sollte der Name doppelt vorkommen, so können wir wieder keine Aussage treffen.

Um den zusammengesetzten Namen eines Angestellten in Vor- und Nachname zu trennen, haben wir eine Prozedur geschrieben, die mittels einer Substring- und charAt-Funktion den Namen korrekt trennt. Als Format haben wir hierfür „<<Name>>, <<Vorname>>“ angegeben. Es erfordert hierfür eine Prozedur, da eine Funktion nur einen Rückgabewert hat, wir aber zwei Werte brauchen. Um das zu erreichen haben wir die Prozedur mit drei Parametern versehen, dem zusammengesetzten Namen als Input, eine Output-Variable in der der Vorname gespeichert wird und gleiches für den Nachnamen.

Um das Alter einer Person zu berechnen haben wir eine bzw. zwei Funktionen geschrieben, die das Geburtsdatum in der jeweiligen Form (YY/MM/DD für Angestellte, MM.YY für Arbeiter) übergeben bekommt. Zunächst werden (Tag,) Monat und Jahr, wie bei der Prozedur zur Namenstrennung, getrennt und dann wird mit den separierten Werten weitergerechnet.

Hier unser Algorithmus in Pseudocode:

IF ((aktuelles Jahr - Geburtsjahr) <= 0)

Alter = Differenz + 100; // Wurde im letzten Jahrhundert geboren

ELSE

Alter = Differenz; // Dieses Jahrhundert geboren

IF ((aktueller Monat - Geburtsmonat) < 0)

Alter -= 1; // Hatte dieses Jahr noch nicht Geburtstag 🡪 abziehen

ELSE IF (Differenz == 0) // Hat diesen Monat Geburtstag

🡪 Tag berücksichtigen wenn möglich

IF ((aktueller Tag - Geburtstag) < 0)

Alter -= 1; // Hatte dieses Jahr noch nicht Geburtstag

🡪 abziehen

RETURN Alter;

Um das korrekte Jahreseinkommen zu berechnen, haben wir erst nach Möglichkeiten gesucht, den Stundenlohn adäquat umzurechnen, da uns nicht wirklich absolut korrekt vorgekommen sind. Als Quelle haben wir Arbeits-rechte.de genommen, da die Webseite einen seriößen Eindruck machte. Diese berechnen die Arbeitstage pro Monat unter <https://www.arbeitsrechte.de/arbeitstage-pro-monat/> wie folgt:

Beispiel für den Normalfall (8h/d, 5d/w):

Um das korrekte Jahreseinkommen aus dem Stundenlohn zu berechnen, nehmen wir o. g. Formel, multiplizieren sie mit 12 Monaten und multiplizieren das mit dem Stundenlohn.

Um das korrekte Jahreseinkommen aus dem Monatsgehalt zu berechnen, nehmen wir einfach diesen und multiplizieren ihn mit 12 Monaten.

Nun haben wir schon fast alle Werte für die Zieltabelle (Nachname, Vorname, Alter, Geschlecht und Jahreseinkommen), alles was noch fehlt sind Personalnummer und Jobcode.

Wir nutzen einen Jobcode (und dazu eine Mapping-Tabelle, die den Code auf eine Berufsbezeichnung mappt), da wir davon ausgehen können, dass einige Berufsbezeichnungen mehrfach vorkommen werden und wir so Speicherplatz sparen können. Wenn wir einen (neuen) Angestellten zu unserem Data Warehouse hinzufügen wollen, prüfen wir, ob es bereits schon einen Code für dessen Berufsbezeichnung gibt, wenn nicht erstellen wir einen 5-stelligen per Zufall generierten Code. Dabei wird auch darauf geachtet, dass dieser Code nicht bereits existiert, sollte das passieren, wird das Prozedere solange wiederholt.

Ähnliches gilt für die Personalnummer. Wenn wir einen neuen Angestellten oder Arbeite hinzufügen wollen prüfen wir, ob dieser bereits in unserem Warehouse ist und wenn nicht, dann generieren wir einen 6-stelligen per Zufall generierten Code. Wie auch schon bei den Jobcodes wird auch hier geprüft, ob diese Personalnummer bereits existiert.

Nun stellte sich uns noch die Frage, wie wir prüfen können, ob es einen Angestellten oder Arbeiter bereits in unserem Data Warehouse gibt. Hierzu haben wir eine weitere Hilfstabelle erstellt, die die Personalnummer, den Namen der Quelltabelle und den alten Primärschlüssel speichert, damit wir den Angestellten / Arbeiter später auch identifizieren können. Beim Angestellten haben wir eine Angestelltennummer, die wir benutzen können, bei den Arbeitern wiederum haben wir das nicht, wir haben den Vor- und Zunamen zur Identifikation genutzt.

Halten wir fest. Wir haben folgende drei Hilfstabellen:

* Eine Tabelle, die einen Vornamen auf einem Geschlecht mappt
* Eine Tabelle, die eine ID auf eine Berufsbezeichnung mappt
* Eine Tabelle, die festhält, ob wir einen Eintrag schon im Data Warehouse haben

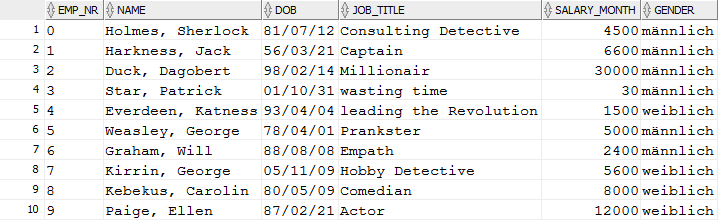
Dazu noch folgende Annahmen:

* Name und Vorname kommt in der Kombination bei einem Arbeiter nur einmal vor
* Arbeiter können 1 Jahr und maximal 100 Jahre alt sein
* Ein Arbeiter arbeite 172 Stunden im Monat, es gibt kein Weihnachtsgehalt
* Es werden zuerst Angestellte und dann Arbeiter hinzugefügt

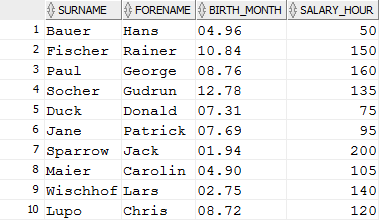
Anbei noch einige Screendumps der Tabellen, um die Korrektheit des Programms zu zeigen:

Vor dem Zusammenführen der Tabellen:

Angestellten-Tabelle:

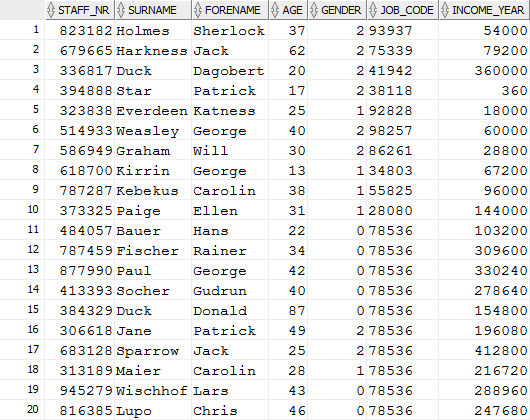


Arbeiter-Tabelle:



Nun wird die Prozedur zur Zusammenführung ausgeführt.

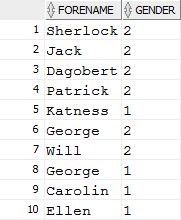
Die Personal-Tabelle sieht wie folgt aus:



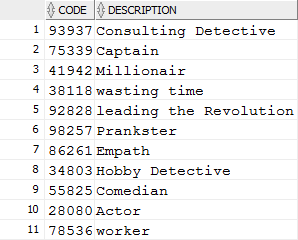
Man beachte, dass George Paul kein Geschlecht zugewiesen wurde, da „George“ einmal als weiblich und einmal als männlich in der Geschlechter-Tabelle steht.

Die Hilfstabellen haben folgende Einträge:

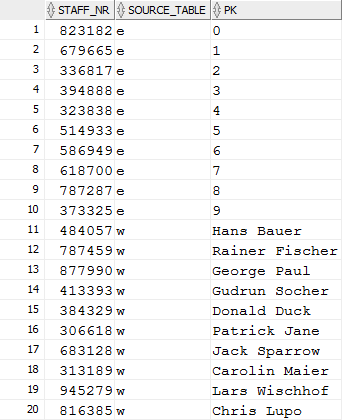
Geschlechts-Mapping-Tabelle:



JobCode-Tabelle:



Identifizierungs-Tabelle:



Nun ändern wir einige Datensätze ab, fügen neue hinzu und führen die Prozedur erneut aus.

Neu:

2 Angestellte (Hannibal Lecter (Angestellter 10) und John Watson (Angestellter 11))

1 Arbeiter (Will Smith, geboren im September 68, hat einen Stundenlohn von 125($))

Verändert:

2 Angestellte:

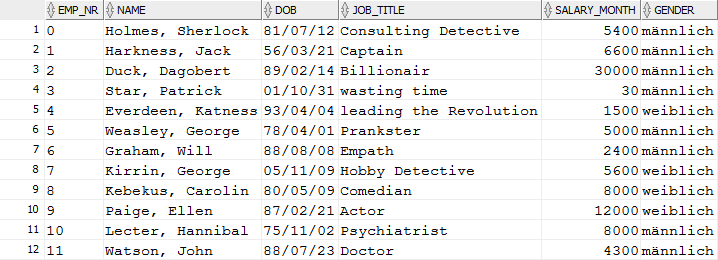
* Dagobert Duck, ist nun Milliardär und nicht mehr Millionär, außerdem wurde das Geburtsdatum von 14.02.98 auf 14.02.89 geändert (Zahlendreher)
* Sherlock Holmes, verdient nun 5.400($) und nicht mehr 4.500($)

1 Arbeiter:

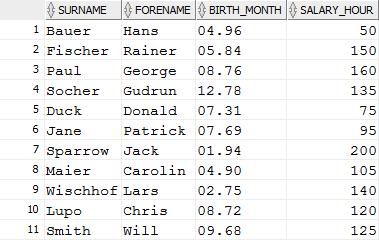
* Rainer Fischer, dieser hat nicht mehr im Oktober sondern im Mai Geburtstag

Nach dem Update:

Angestellten-Tabelle:

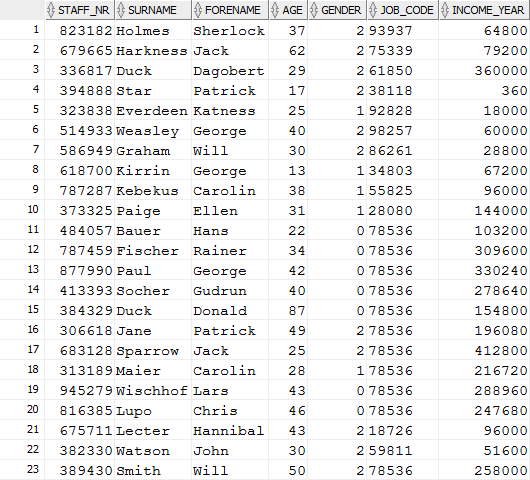


Arbeiter-Tabelle:



Und nun wird die Prozedur erneut ausgeführt.

Die Personal-Tabelle sieht nun wie folgt aus:

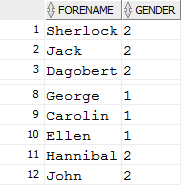
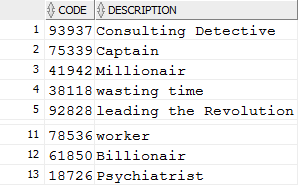
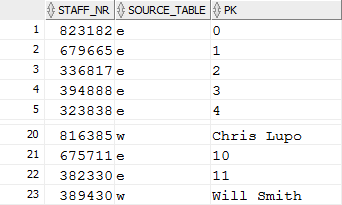


Man beachte, dass

* das Jahreseinkommen von Sherlock Holmes nun 64.800 und nicht mehr 54.000 ist.
* Dagobert Duck einen anderen Jobcode hat und er nicht mehr 20 sondern 29 ist.
* Will Smith als männlich eingetragen ist, da „Will“ in der Gender-Tabelle ist.
* das Alter von Rainer Fischer auch weiterhin bei 34 ist, da gerade Januar ist.
* die bereits existierenden Einträge nicht doppelt in unserer Tabelle sind.

Änderungen in den Hilfstabellen:

Geschlechter: Job-Codes: Identifizierung:

# Quellcode

**Die SQL-Files und die Doku finden Sie zusätzlich auf** [**GitHub.com/AnyaW/DBS2**](https://github.com/AnyaW/dbs2)**.**

**Min-Max-Skalierung:**

CREATE TABLE People (

person\_id INT NOT NULL,

person\_name VARCHAR(63),

person\_age INT,

person\_place VARCHAR(63),

CONSTRAINT pers\_id PRIMARY KEY (person\_id)

);

INSERT INTO People VALUES(0, 'Hans', 34, 'Bonn');

INSERT INTO People VALUES(1, 'Werner', 18, 'Berlin');

INSERT INTO People VALUES(2, 'Jürgen', 74, 'München');

INSERT INTO People VALUES(3, 'Luis', 25, 'Augsburg');

INSERT INTO People VALUES(4, 'Nicklas', 22, 'Regensburg');

INSERT INTO People VALUES(5, 'Jens', 49, 'Prag');

INSERT INTO People VALUES(6, 'Marvin', 17, 'Cottbus');

INSERT INTO People VALUES(7, 'Sophia', 22, 'München');

INSERT INTO People VALUES(8, 'Günther', 63, 'Ulm');

INSERT INTO People VALUES(9, 'Sebastian', 19, 'Ingolstadt');

CREATE OR REPLACE FUNCTION min\_max\_scale(v IN INT, new\_min IN INT, new\_max IN INT, old\_min IN INT, old\_max IN INT)

RETURN INT

IS new\_v INT;

BEGIN

new\_v := ((v - old\_min) / (old\_max - old\_min)) \* (new\_max - new\_min) + new\_min; -- min max impl

RETURN(new\_v);

END min\_max\_scale;

/

CREATE OR REPLACE PROCEDURE MinMax\_Scaling (new\_min INT, new\_max INT) IS

old\_min INT;

old\_max INT;

new\_age INT;

BEGIN

SELECT MIN(person\_age) as min\_age, MAX(person\_age) as max\_age INTO old\_min, old\_max FROM People; -- get min and max

FOR person IN (SELECT \* FROM People) LOOP

new\_age := min\_max\_scale(person.person\_age, new\_min, new\_max, old\_min, old\_max);

UPDATE People SET person\_age = new\_age WHERE person\_id = person.person\_id;

END LOOP;

END;

/

EXECUTE MinMax\_Scaling(10, 50);

**Data Warehouse:**

Tables:

CREATE TABLE Employee (

emp\_nr VARCHAR(9) NOT NULL,

name VARCHAR(31), -- surname, forname

dob VARCHAR(8), -- yy/mm/dd

job\_title VARCHAR(63),

salary\_month DOUBLE PRECISION,

gender VARCHAR(9), -- männlich / weiblich

CONSTRAINT employee\_nr PRIMARY KEY (emp\_nr)

);

CREATE TABLE Worker (

surname VARCHAR(31),

forename VARCHAR(31),

birth\_month VARCHAR(5), -- mm.yy

salary\_hour DOUBLE PRECISION

);

CREATE TABLE Staff (

staff\_nr INTEGER NOT NULL,

surname VARCHAR(31),

forename VARCHAR(31),

age INT,

gender INT, -- 0 = unknown, 1 = female, 2 = male

job\_code VARCHAR(7),

income\_year DOUBLE PRECISION,

CONSTRAINT s\_nr PRIMARY KEY (staff\_nr)

);

CREATE TABLE Jobcode (

code INTEGER PRIMARY KEY,

description VARCHAR(63)

);

CREATE TABLE Gender (

forename VARCHAR(31),

gender VARCHAR(1), -- 1 = female, 2 = male

CONSTRAINT gender\_entry PRIMARY KEY (forename, gender)

);

CREATE TABLE Identifier (

staff\_nr INTEGER PRIMARY KEY,

source\_table VARCHAR(1), -- e = Employee, w = Worker

pk VARCHAR(63) -- Employee: emp\_nr, Worker: Forename + Surname

);

Inserts:

INSERT INTO Employee VALUES ('0', 'Holmes, Sherlock', '81/07/12', 'Consulting Detective', 4500.00, 'männlich');

INSERT INTO Employee VALUES ('1', 'Harkness, Jack', '56/03/21', 'Captain', 6600.00, 'männlich');

INSERT INTO Employee VALUES ('2', 'Duck, Dagobert', '98/02/14', 'Millionair', 30000.00, 'männlich');

INSERT INTO Employee VALUES ('3', 'Star, Patrick', '01/10/31', 'wasting time', 30.00, 'männlich');

INSERT INTO Employee VALUES ('4', 'Everdeen, Katness', '93/04/04', 'leading the Revolution', 1500.00, 'weiblich');

INSERT INTO Employee VALUES ('5', 'Weasley, George', '78/04/01', 'Prankster', 5000.00, 'männlich');

INSERT INTO Employee VALUES ('6', 'Graham, Will', '88/08/08', 'Empath', 2400.00, 'männlich');

INSERT INTO Employee VALUES ('7', 'Kirrin, George', '05/11/09', 'Hobby Detective', 5600.00, 'weiblich');

INSERT INTO Employee VALUES ('8', 'Kebekus, Carolin', '80/05/09', 'Comedian', 8000.00, 'weiblich');

INSERT INTO Employee VALUES ('9', 'Paige, Ellen', '87/02/21', 'Actor', 12000.00, 'weiblich');

INSERT INTO Worker VALUES ('Bauer', 'Hans', '04.96', 50.0);

INSERT INTO Worker VALUES ('Fischer', 'Rainer', '10.84', 150.0);

INSERT INTO Worker VALUES ('Paul', 'George', '08.76', 160.0);

INSERT INTO Worker VALUES ('Socher', 'Gudrun', '12.78', 135.0);

INSERT INTO Worker VALUES ('Duck', 'Donald', '07.31', 75.0);

INSERT INTO Worker VALUES ('Jane', 'Patrick', '07.69', 95.0);

INSERT INTO Worker VALUES ('Sparrow', 'Jack', '01.94', 200.0);

INSERT INTO Worker VALUES ('Maier', 'Carolin', '04.90', 105.0);

INSERT INTO Worker VALUES ('Wischhof', 'Lars', '02.75', 140.0);

INSERT INTO Worker VALUES ('Lupo', 'Chris', '08.72', 120.0);

Merge:

SET SERVEROUTPUT ON;

-- private function

CREATE OR REPLACE FUNCTION generate\_id(digits IN INTEGER)

RETURN INTEGER

IS

new\_id INTEGER;

max\_nr INTEGER := 9;

id\_str VARCHAR(10);

cur\_nr INTEGER;

starts\_with\_zero BOOLEAN;

BEGIN

FOR i in 1..digits LOOP

cur\_nr := dbms\_random.value(0, max\_nr);

id\_str := CONCAT(id\_str, cur\_nr);

END LOOP;

starts\_with\_zero := INSTR(id\_str, '0') = 1;

IF starts\_with\_zero THEN

id\_str := CONCAT(id\_str, '9');

END IF;

new\_id := TO\_NUMBER(id\_str);

RETURN new\_id;

END;

/

CREATE OR REPLACE PROCEDURE get\_staffnr(identify IN VARCHAR, source IN VARCHAR, staffnr OUT INTEGER, exist OUT BOOLEAN)

IS

results INTEGER;

BEGIN

SELECT COUNT(\*) INTO results FROM Identifier WHERE pk = identify AND source\_table = source;

IF results = 0 THEN

staffnr := -1;

WHILE staffnr = -1 OR results != 0 LOOP

staffnr := generate\_id(6);

SELECT COUNT(\*) INTO results FROM Identifier WHERE staff\_nr = staffnr;

END LOOP;

exist := FALSE;

INSERT INTO Identifier VALUES (staffnr, source, identify);

ELSE

SELECT src.staff\_nr INTO staffnr FROM Identifier src WHERE pk = identify;

exist := TRUE;

END IF;

END;

/

CREATE OR REPLACE PROCEDURE split\_name(staff\_name IN VARCHAR, surname IN OUT VARCHAR, forename IN OUT VARCHAR)

IS

BEGIN

surname := '';

forename := '';

SELECT SUBSTR(staff\_name, 1, INSTR(staff\_name, ',', 1, 1) - 1),

SUBSTR(staff\_name, INSTR(staff\_name, ',', 1, 1) + 2)

INTO surname, forename

FROM DUAL;

END;

/

CREATE OR REPLACE FUNCTION concat\_name(forename IN VARCHAR, surname IN VARCHAR)

RETURN VARCHAR

IS

fullname VARCHAR(64);

BEGIN

fullname := CONCAT(CONCAT(forename, ' '), surname);

RETURN fullname;

END;

/

CREATE OR REPLACE PROCEDURE split\_date\_emp(dt IN VARCHAR, yy IN OUT INTEGER, mm IN OUT INTEGER, dd IN OUT INTEGER)

IS

BEGIN

yy := 0;

mm := 0;

dd := 0;

-- SUBSTR takes a String, the position of the Beginning and optional the length

-- INSTR takes a haystack, a needed, the starting index and the occurrence

-- String from position 1 (first char) going to first occurrence of '/' - 1 = year

SELECT TO\_NUMBER(SUBSTR(dt, 1, INSTR(dt, '/', 1, 1) - 1)),

-- String from the first occurrence of '/' + 1 going for the amount of chars

-- between the second and the first occurrence - 1 to remove the slash = month

TO\_NUMBER(SUBSTR(dt, INSTR(dt, '/', 1, 1) + 1, INSTR(dt, '/', 1, 2) - INSTR(dt, '/', 1, 1) - 1)),

-- String from the second occurrence of '/' going to the end of the String = day

TO\_NUMBER(SUBSTR(dt, INSTR(dt, '/', 1, 2) + 1))

INTO yy, mm, dd

FROM DUAL;

END;

/

CREATE OR REPLACE FUNCTION dob\_age\_emp(date\_dob IN VARCHAR)

RETURN INTEGER

IS

age INTEGER;

year\_dob INTEGER;

month\_dob INTEGER;

day\_dob INTEGER;

year\_cur INTEGER;

month\_cur INTEGER;

day\_cur INTEGER;

date\_cur VARCHAR(8);

year\_diff INTEGER;

month\_diff INTEGER;

day\_diff INTEGER;

BEGIN

-- Get Current Time

SELECT to\_char(sysdate, 'yy/mm/dd') INTO date\_cur FROM DUAL;

-- Split Date into day, month and year

split\_date\_emp(date\_dob, year\_dob, month\_dob, day\_dob);

split\_date\_emp(date\_cur, year\_cur, month\_cur, day\_cur);

age := 0;

-- diff not positive? Add 100, else it's the difference.

year\_diff := year\_cur - year\_dob;

IF (year\_diff <= 0) THEN

age := year\_diff + 100;

ELSE

age := year\_diff;

END IF;

-- diff negative? Didn't have his birthday this year so remove it.

month\_diff := month\_cur - month\_dob;

IF (month\_diff < 0) THEN

age := age - 1;

ELSE

-- same month? We need to check the day then.

IF (month\_diff = 0) THEN

day\_diff := day\_cur - day\_dob;

-- diff negative? Didn't have his birthday this year so remove it.

IF (day\_diff < 0) THEN

age := age - 1;

END IF;

END IF;

END IF;

RETURN age;

END;

/

CREATE OR REPLACE PROCEDURE split\_date\_wor(dt IN VARCHAR, yy IN OUT INTEGER, mm IN OUT INTEGER) IS

BEGIN

mm := 0;

yy := 0;

-- SUBSTR takes a String, the position of the Beginning and optional the length

-- INSTR takes a haystack, a needed, the starting index and the occurrence

-- String from position 1 (first char) going to first occurrence of '.' - 1 = month

SELECT TO\_NUMBER(SUBSTR(dt, 1, INSTR(dt, '.', 1, 1) - 1)),

-- String from the first occurrence of '.' + 1 going until the end = year

TO\_NUMBER(SUBSTR(dt, INSTR(dt, '.', 1, 1) + 1))

INTO mm, yy

FROM DUAL;

END;

/

CREATE OR REPLACE FUNCTION dob\_age\_wor(date\_dob IN VARCHAR)

RETURN INTEGER

IS

year\_dob INTEGER;

month\_dob INTEGER;

year\_cur INTEGER;

month\_cur INTEGER;

date\_cur VARCHAR(8);

year\_diff INTEGER;

month\_diff INTEGER;

age INTEGER;

BEGIN

-- Get Current Time

SELECT to\_char(sysdate, 'mm.yy') INTO date\_cur FROM DUAL;

-- Split Date into day, month and year

split\_date\_wor(date\_dob, year\_dob, month\_dob);

split\_date\_wor(date\_cur, year\_cur, month\_cur);

age := 0;

-- diff not positive? Add 100, else it's correct already.

year\_diff := year\_cur - year\_dob;

IF (year\_diff <= 0) THEN

age := year\_diff + 100;

END IF;

-- diff negative? Didn't have his birthday yet so remove it.

month\_diff := month\_cur - month\_dob;

IF (month\_diff < 0) THEN

age := age - 1;

END IF;

RETURN age;

END;

/

-- this is only for worker

CREATE OR REPLACE FUNCTION gender\_mapping(fname IN VARCHAR)

RETURN VARCHAR IS

answer INTEGER;

gen VARCHAR(1);

BEGIN

-- if name only once in TABLE then use this gender -> 1, 2

-- else gender is unknown -> 0

SELECT COUNT(\*)

INTO answer

FROM Gender g

WHERE g.forename = fname;

IF answer = 1 THEN

SELECT g.gender INTO gen

FROM Gender g

WHERE g.forename = fname;

ELSE

gen := '0';

END IF;

RETURN gen;

END;

/

-- this is only for employees

CREATE OR REPLACE FUNCTION gender\_evaluation(fname IN VARCHAR, gen IN VARCHAR)

RETURN VARCHAR IS

gen\_nr VARCHAR(1);

answer INTEGER;

BEGIN

-- if e then map m, w to 2, 1

-- check if already in gender table

-- if false then insert into TABLE Gender

-- return 2 or 1

IF gen = 'weiblich' THEN

gen\_nr := '1';

ELSE

IF gen = 'männlich' THEN

gen\_nr := '2';

END IF;

END IF;

SELECT COUNT(\*)

INTO answer

FROM Gender g

WHERE g.forename = fname AND g.gender = gen\_nr;

IF answer = 0 THEN

INSERT INTO Gender VALUES (fname, gen\_nr);

COMMIT;

END IF;

RETURN gen\_nr;

END;

/

CREATE OR REPLACE FUNCTION get\_jobcode(job\_title IN VARCHAR)

RETURN INTEGER

IS

job\_code INTEGER;

results INTEGER;

BEGIN

SELECT COUNT(\*) INTO results FROM Jobcode WHERE description = job\_title;

IF results = 0 THEN

job\_code := -1;

WHILE job\_code = -1 OR results != 0 LOOP

job\_code := generate\_id(5);

SELECT COUNT(\*) INTO results FROM Jobcode WHERE code = job\_code;

END LOOP;

INSERT INTO Jobcode VALUES (job\_code, job\_title);

ELSE

SELECT code INTO job\_code FROM Jobcode WHERE description = job\_title;

END IF;

RETURN job\_code;

END;

/

CREATE OR REPLACE FUNCTION annual\_income(income IN DOUBLE PRECISION, source IN VARCHAR)

RETURN DOUBLE PRECISION

IS

year\_income DOUBLE PRECISION;

BEGIN

IF (source = 'e') THEN

year\_income := income \* 12;

ELSE

IF (source = 'w') THEN

year\_income := income \* 12 \* (8 \* 5 \* 4.3);

END IF;

END IF;

RETURN year\_income;

END;

/

CREATE OR REPLACE PROCEDURE merge\_employees

IS

-- var

entry\_exist BOOLEAN;

-- to save

staffnr INTEGER;

sname VARCHAR(31);

fname VARCHAR(31);

a INTEGER;

jobcode INTEGER;

income INTEGER;

gen VARCHAR(1);

BEGIN

FOR employee IN (SELECT \* FROM Employee) LOOP

-- procedures (multiple output values)

get\_staffnr(employee.emp\_nr, 'e', staffnr, entry\_exist);

split\_name(employee.name, sname, fname); -- split name

-- functions

a := dob\_age\_emp(employee.dob); -- calc age

gen := gender\_evaluation(fname, employee.gender);

jobcode := get\_jobcode(employee.job\_title);

income := annual\_income(employee.salary\_month, 'e');

IF entry\_exist THEN

UPDATE Staff s

SET s.surname = sname, s.forename = fname, s.age = a, s.gender = gen, s.job\_code = jobcode, s.income\_year = income

WHERE s.staff\_nr = staffnr;

ELSE

INSERT INTO Staff

VALUES(staffnr, sname, fname, a, gen, jobcode, income);

END IF;

END LOOP;

END;

/

CREATE OR REPLACE PROCEDURE merge\_worker

IS

identify VARCHAR(64);

exist BOOLEAN;

staffnr INTEGER;

a INTEGER;

gen VARCHAR(1);

jobcode INTEGER;

income DOUBLE PRECISION;

BEGIN

FOR worker IN (SELECT \* FROM Worker) LOOP

identify := concat\_name(worker.forename, worker.surname);

get\_staffnr(identify, 'w', staffnr, exist);

a := dob\_age\_wor(worker.birth\_month);

gen := gender\_mapping(worker.forename);

jobcode := get\_jobcode('worker');

income := annual\_income(worker.salary\_hour, 'w');

IF exist THEN

UPDATE Staff s

SET s.age = a, s.gender = gen, s.job\_code = jobcode, s.income\_year = income

WHERE s.staff\_nr = staffnr;

ELSE

INSERT INTO Staff

VALUES (staffnr, worker.surname, worker.forename, a, gen, jobcode, income);

END IF;

END LOOP;

END;

/

CREATE OR REPLACE PROCEDURE merge\_staff

IS

BEGIN

merge\_employees();

merge\_worker();

END;

/

EXECUTE merge\_staff;

Updates:

-- Update Tables after first execute

-- Employees

INSERT INTO Employee VALUES ('10', 'Lecter, Hannibal', '75/11/02', 'Psychiatrist', 8000.00, 'männlich');

INSERT INTO Employee VALUES ('11', 'Watson, John', '88/07/23', 'Doctor', 4300.00, 'männlich');

-- dob and job title

UPDATE Employee e

SET e.name = 'Duck, Dagobert', e.dob = '89/02/14',

e.job\_title = 'Billionair',

e.salary\_month = 30000.00, e.gender = 'männlich'

WHERE e.emp\_nr = '2';

-- salery

UPDATE Employee e

SET e.name = 'Holmes, Sherlock', e.dob = '81/07/12',

e.job\_title = 'Consulting Detective',

e.salary\_month = 5400.00, e.gender = 'männlich'

WHERE e.emp\_nr = '0';

-- Worker

INSERT INTO Worker VALUES ('Smith', 'Will', '09.68', 125.0);

-- dob

UPDATE Worker w

SET w.birth\_month = '05.84', w.salary\_hour = 150.0

WHERE w.surname = 'Fischer' AND w.forename = 'Rainer';