

# PL/SQL H2

## Functies

HOGESCHOOL 



# Ingebouwde functies

- Bekend vanuit de lessen SQL

Vb     SELECT sysdate FROM dual;  
          SELECT SUBSTR(last\_name, 1, 4)  
          FROM employees;

Er bestaan functies met of zonder parameters.  
Een functie geeft altijd 1 resultaat terug.

# Wat is een functie?

- object (bestaat uit statements en PL/SQL-constructies) met een naam
- wordt bewaard in de DB
- code op 1 plaats definiëren en op meerdere plaatsen gebruiken
- retourneert een waarde

# Syntax voor de creatie van een functie

Het PL/SQL blok moet minstens 1 RETURN statement bevatten.

```
CREATE [OR REPLACE] FUNCTION function_name
  [(parameter1 [mode1] datatype1, ...)]
RETURN datatype IS|AS
  [local_variable_declarations; ...]
BEGIN
  -- actions;
  RETURN expression;
END [function_name];
```

# Voorbeeld: functie zonder parameters

```
CREATE OR REPLACE FUNCTION sysdate2
```

```
RETURN VARCHAR2
```

Return-type: geen lengte!!

```
AS
```

```
    v_tekst    VARCHAR2(30);
```

Declaratie van variabelen

```
BEGIN
```

Statements

```
    v_tekst := TO_CHAR(sysdate, 'fmDay, dd month yyyy');
```

```
    RETURN v_tekst;
```

```
END;
```

```
/
```

# PL/SQL Syntax

- Elke instructie eindigt met ;
- Een toekenning gebeurt door :=
- In een PL/SQL blok kan je andere functies oproepen, je hoeft hiervoor geen SELECT-statement te gebruiken (behalve bij groepsfuncties en DECODE).

Vb.     v\_lengte     := LENGTH(v\_naam);  
        v\_rest      := MOD(v\_lengte, 2);  
        v\_hoofd     := UPPER(v\_naam);

- Commentaar toevoegen aan je code:
  - 1 regel: -- dit is 1 lijn commentaar
  - meerdere regels: /\* commentaar over meerdere lijnen \*/

# Opmerkingen bij functies

- Een functie heeft (in tegenstelling tot een gewoon PL/SQL blok) altijd een naam. Een gewoon PL/SQL blok noemt men ook wel **een ANONIEM blok**.
- Bij het RETURN-type mag geen lengte meegegeven worden.
- Tussen IS/AS en BEGIN kan je variabelen declareren (zie verder).
- In de body van de functie → minstens 1 return-commando met daarachter de waarde die wordt teruggegeven (van het type zoals hoger beschreven!)
- Functies kunnen van een **argumentenlijst** worden voorzien, door deze argumenten of **parameters** wordt de flexibiliteit vergroot (zie verder).

## / → creatie functie

- de broncode wordt altijd in de data dictionary opgeslagen
- als foutloze code: gecompileerde versie → databank
- als code met fouten:  
Melding: ``created with compilation errors'`.`

Hoe fouten opvragen: `show errors`



# Functie gebruiken

- Vanuit een andere functie:

in het BEGIN-blok: `v_datum := sysdate2;`

- Vanuit een SQL-instructie:

**SQL>** `SELECT sysdate2 FROM dual;`

# Declaratie en initialisatie van PL/SQL variabelen

Syntax:

```
identifier [CONSTANT] datatype [NOT NULL]
           [:= | DEFAULT expr];
```

Voorbeelden:

```
v_hiredate    DATE;
v_deptno      NUMBER(2) NOT NULL := 10;
v_location    VARCHAR2(13) := 'Atlanta';
v_comm        CONSTANT NUMBER := 1400;
```

# Naamgeving variabelen

- Moet beginnen met een letter
  - Mag letters en getallen bevatten
  - Mag bevatten: dollar teken (\$), underscore, pond teken (£)
  - Maximale lengte is 30 tekens
  - Geen gereserveerde woorden
- 
- Opmerking: naam begint met v\_

# Belangrijkste Scalar Data Types

- CHAR [(maximum\_lengte)]
- VARCHAR2 (maximum\_lengte)
- LONG
- NUMBER [(precisie, schaal)]
- BINARY\_INTEGER
- PLS\_INTEGER
- BOOLEAN
- BINARY\_FLOAT: sneller, maar minder precies
- BINARY\_DOUBLE
- DATE
- TIMESTAMP

# Declaratie: %TYPE

verwijzen naar het datatype van een andere variabele

- datatype eerder beschreven variabele
- datatype kolom uit databank

Syntax:

```
identifier    table.column_name%TYPE;  
identifier    other_variable%TYPE;
```

Vb.

v_getal	NUMBER(4,1);
v_getal2	v_getal%TYPE;
v_mndsal	employees.salary%TYPE;

→ dit best doen voor variabelen die hun waarde uit de database krijgen (zie later)

→ enkel het datatype wordt overgenomen, geen default-waarde

# Declaratie: default-waarde

- zonder DEFAULT-waarde: NULL
- bij beschrijving NOT NULL of CONSTANT → DEFAULT-waarde verplicht

Vb

v_account	NUMBER(11)	NOT NULL	:= 1200000;
v_bonus	NUMBER(2)	DEFAULT 0;	
v_naam1	VARCHAR2(20)	DEFAULT 'X';	
v_naam2	v_naam1%TYPE	DEFAULT 'Y';	
v_vandaag	DATE	DEFAULT SYSDATE;	
v_gisteren	DATE	DEFAULT SYSDATE - 1;	
v_max	CONSTANT NUMBER(4)	:= 5000;	

# Operatoren

- rekenkundig

\*\* \* / + -

- alfanumeriek

||

- vergelijking

= != <> < <= > >=

IS NULL

LIKE

BETWEEN

IN

- logisch

AND OR NOT

# Oefening 1



## Voorbeeld: functie met parameters

```
CREATE OR REPLACE FUNCTION fulldate  
(p_date IN DATE)  
    RETURN VARCHAR2  
  
AS  
  
BEGIN  
    RETURN TO_CHAR(p_date, 'fmDay, dd month yyyy');  
  
END;
```

# Parameterlijst

(p\_tekst  
p\_sal  
...)

naam

p\_...

varchar2,  
employees.salary%TYPE,

datatype

(geen lengte!!)

Meerdere parameters zijn gescheiden door een komma

# Functie gebruiken

- Vanuit een andere functie:

in het BEGIN-blok: `v_tekst := fulldate(v_datum);`

- Vanuit een SQL-instructie:

**SQL>** `SELECT fulldate(hire_date) FROM employees;`

# Oefening 2

# Oefening 3a

# Voorwaardelijke uitvoering: IF

Syntax:

```
IF condition THEN
    statements;
[ELSIF condition THEN
    statements;]
[ELSE
    statements;]
END IF;
```

# Voorwaardelijke uitvoering

## Voorbeeld

```
IF v_leeftijd > 60 THEN
    v_categorie := 'senior';
ELSE
    v_categorie := 'middelbaar';
END IF;
```

# Voorwaardelijke uitvoering: geneste IF

## Voorbeeld

```
IF v_leeftijd > 60 THEN
    v_categorie := 'senior';
ELSE
    IF v_leeftijd > 35 THEN
        v_categorie := 'middelbaar';
    ELSE
        v_categorie := 'jong';
    END IF;
END IF;
```

# Voorwaardelijke uitvoering: ELSIF

## Voorbeeld

```
IF v_leeftijd > 60 THEN
    v_categorie := 'senior';
ELSIF v_leeftijd > 35 THEN
    v_categorie := 'middelbaar';
ELSE
    v_categorie := 'jong';
END IF;
```



# Voorwaardelijke uitvoering: AND en OR

## Voorbeeld1

```
IF v_leeftijd > 60 AND status = 'niet werkend' THEN  
    v_categorie := 'gepensioneerde senior';  
END IF;
```

## Voorbeeld2

```
IF v_leeftijd < 18 OR status = 'student' THEN  
    v_belastingen := 0;  
END IF;
```

**Oefening 3b**

**Oefening 4**

**Oefening 5**

# SELECT Statements in PL/SQL

- Gegevens uit de databank ophalen met een `SELECT` statement.
- Syntax:

```
SELECT  select_list
INTO    {variable_name[, variable_name]...
        | record_name}
FROM    table
[WHERE  condition];
```

- De INTO-clause is verplicht!
- Een query MOET 1 rij ophalen! → WHERE-clausule

# SELECT Statements in PL/SQL: voorbeeld

```
CREATE OR REPLACE FUNCTION get_aantal_dienstjaren
(p_emp_id          employees.employee_id%TYPE)
RETURN NUMBER
AS
    v_hire_date          employees.hire_date%TYPE;
    v_aantal_jaren_dienst NUMBER(2);
BEGIN
    SELECT hire_date
    INTO v_hire_date
    FROM employees
    WHERE employee_id = p_emp_id;
    v_aantal_jaren_dienst := TRUNC(MONTHS_BETWEEN(sysdate, v_hire_date)/12);
    RETURN v_aantal_jaren_dienst;
END;
```

# SELECT Statements in PL/SQL: opmerkingen

- volledige syntax van select statement kan gebruikt worden, incl. WHERE, GROUP BY en HAVING
- meer dan 1 variabele kan gevuld worden  
aantal expr. in SELECT = aantal variabelen

## Voorbeeld

```
SELECT department_name, SUM(salary)
INTO v_dep_name, v_som
FROM employees JOIN departments USING(department_id)
WHERE department_id = 80;
```

# Programming Guidelines

- Maak je code leesbaarder en beter onderhoudbaar:
  - Kies duidelijke namen voor je variabelen
  - Zorg voor een duidelijke inspringing

```
BEGIN
  IF x = 0 THEN
    y:=1;
  END IF;
END;
/
```

```
...
AS
  v_deptno          NUMBER(4);
  v_location_id     NUMBER(4);
BEGIN
  SELECT  department_id,
          location_id
  INTO    v_deptno,
          v_location_id
  FROM    departments
  WHERE   department_name
          = 'Sales';

...
END;
/
```

# Functies verwijderen

- Syntax:

```
DROP FUNCTION function_name
```

- Voorbeeld: `DROP FUNCTION get_jaarsal;`
  - Alle privileges betreffende de functie worden mee verwijderd.
  - De `CREATE OR REPLACE` syntax is equivalent aan het verwijderen en opnieuw creëren van de functie. Toegekende privileges i.v.m. de functie blijven bestaan als deze syntax gebruikt wordt.

# Opvragen kenmerken (data dictionary)

`desc naam_functie`

→ Overzicht van invoer- en uitvoerparameters van de functie

## Voorbeeld

```
SQL> desc netto
FUNCTION netto RETURNS VARCHAR2
Argument Name          Type                      In/Out Default?
-----
P_BRUTO                 NUMBER(8,2)              IN
```



# Opvragen kenmerken (data dictionary)

Alle informatie over bestaande PL/SQL functies is bewaard in de databank. Je kan hiervoor gebruik maken van volgende Oracle data dictionary views:

- `USER_OBJECTS`: deze view bevat informatie over ALLE databankobjecten van de eigen user, dus alle zelf-gecreëerde tabellen, indexen, sequences, functies,...
- `USER_SOURCE`: hierin zit de code van bepaalde objecten

# Opvragen kenmerken (data dictionary)

## USER\_OBJECTS

belangrijkste kolommen zijn object\_name, object\_type, created, ...

Voorbeeld om te kijken welke functies aanwezig zijn:

```
SELECT object_name
FROM   user_objects
WHERE  object_type = 'FUNCTION';
```

# Opvragen kenmerken (data dictionary)

## USER\_SOURCE

belangrijkste kolommen zijn name, type, line, text

Voorbeeld om de code van een bestaande functie te bekijken:

```
SELECT text
FROM   user_source
WHERE  name = 'GET_JAARSAL';
```

# Oefeningen