

# PL/SQL H1

## Introductie

HOGESCHOOL 



# Wat is PL/SQL?

- PL/SQL
- Procedural Language/Structured Query Language
- Maakt het mogelijk om SQL (een niet-procedurele 4<sup>de</sup> generatietaal) uit te breiden met elementen uit een procedurele 3<sup>de</sup> generatietaal

# Mogelijkheden SQL

- SQL: nodig om relationele database te maken en te beheren
- 4GL = declaratief



definiëren **wat** we willen, niet **hoe** we tot een resultaat komen

vb. `SELECT * FROM medewerkers;`

# Onmogelijkheden SQL

- gebruik van variabelen  
(subquery = gedeeltelijke opl.)
- records in bepaalde volgorde verwerken
- ontbreken procedurele constructies
  - voorwaarde
  - iteratie

# Mogelijkheden PL/SQL

- gebruik variabelen en constanten
- programmabesturing (iteratie en selectie)
- mogelijkheden om rij per rij-verwerking te kunnen doen
- modulariteit: programma's opdelen in kleinere eenheden
- afschermen van broncode door packages
- foutafhandeling (exceptions)

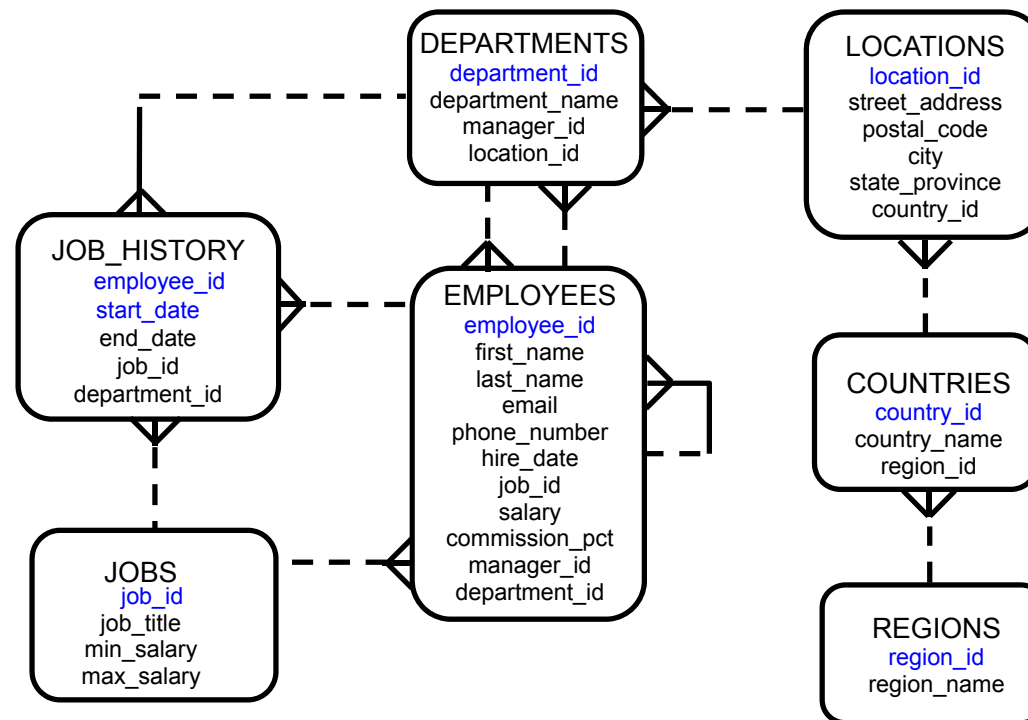
# PL/SQL Programming Environments

- SQL Developer
- SQL\*Plus (idem opleidingsonderdeel Data)
- Jdeveloper

# SQL\*Plus

- Vooraf Oracle XE installeren volgens installatieprocedure op BB
- Gebruikte tabellen: idem opleidingsonderdeel Data

# The Human Resources (HR) Schema





# PL/SQL H2

## Funcities

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;	

## 2.3 Operatoren

- rekenkundige

\*\* \* / + -

- alfanumerieke

||

- vergelijkingen  
(levert TRUE, FALSE of UNK op)

= != <> < <= > >=  
IS NULL  
LIKE  
BETWEEN  
IN

- logische

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;
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

# PL/SQL H3

## Procedures



# Wat is een procedure?

- object (bestaat uit SQL-statements en PL/SQL-constructies) met een naam – ook subprogramma genoemd
- wordt bewaard in de DB
- code op 1 plaats definiëren en op meerdere plaatsen gebruiken
- voert één of meerdere acties uit
- kan aangeroepen(called) worden met 1 of meerdere parameters
- slechts 1 aanroep voor meerdere acties waardoor betere performance

# Syntax voor de creatie van een procedure

```
CREATE [OR REPLACE] PROCEDURE procedure_name  
[(parameter1 [mode] datatype1,  
  parameter2 [mode] datatype2, ...)]  
IS|AS  
  [local_variable_declarations; ...]  
BEGIN  
  -- actions;  
END [procedure_name];
```

# Voorbeeld: procedure zonder parameters

```
CREATE OR REPLACE PROCEDURE show_emp
```

```
IS
```

```
    v_emp            employees.employee_id%type := '100';
```

```
    v_voornaam       employees.first_name%TYPE;
```

```
    v_naam           employees.last_name%TYPE;
```

```
BEGIN
```

```
    SELECT first_name, last_name
```

```
    INTO v_voornaam, v_naam
```

```
    FROM employees
```

```
    WHERE employee_id = v_emp;
```

```
    DBMS_OUTPUT.PUT_LINE(v_voornaam || ' ' || v_naam);
```

```
END show_emp;
```

Let op: als employee\_id 100 niet bestaat in de tabel employees dan zal de procedure afsluiten met de foutmelding: “no data found”



# Afdrukken

- Gebruik maken van ingebouwde package DBMS\_OUTPUT met mogelijke procedures o.a. PUT\_LINE
- Tussen ronde haakjes de af te drukken lijn als 1 geheel meegeven
- Vb. DBMS\_OUTPUT.PUT\_LINE('Je kan tekst en variabelen samenvoegen met concatenatie-teken' || ' alle tekst moet tussen single quotes. Dit moet niet voor variabelen.' || v\_country\_name);
- in SQL\*Plus SET SERVEROUTPUT ON opnemen → best in LOGIN.SQL
- LOGIN.SQL is een script dat telkens bij het opstarten van SQL\*Plus wordt uitgevoerd en waar bepaalde settings kunnen opgenomen worden

## / → creatie procedure

- de broncode wordt in ieder geval 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`

# Voorbeeld: procedure zonder parameters

```
CREATE OR REPLACE PROCEDURE add_ctr  
IS  
    v_country_id          countries.country_id%type := 'FR';  
    v_country_name        countries.country_name%type := 'France';  
    v_region_id           countries.region_id%type := 1;  
  
BEGIN  
    INSERT INTO countries  
    VALUES (v_country_id,v_country_name,v_region_id);  
    DBMS_OUTPUT.PUT_LINE('Er werden ' || SQL%ROWCOUNT || ' rijen  
    toegevoegd in de tabel COUNTRIES');  
END add_ctr;  
/
```

# PL/SQL Syntax

- Elk DML-statement(INSERT, UPDATE, DELETE) kan zonder aanpassingen aan syntax opgenomen worden in PL/SQL
- Let op: een **DML-statement** in PL/SQL geeft geen fout als er 0 of meerdere rijen bewerkt worden
- Een **SELECT-statement** in PL/SQL dat 0 resultaatrijen geeft zal leiden tot de foutmelding 'NO DATA FOUND'
- Een **SELECT-statement** in PL/SQL dat meerdere resultaatrijen geeft zal leiden tot de foutmelding 'EXACT FETCH RETURNS MORE THAN REQUESTED NUMBER OF ROWS'

# Impliciete cursor

Info omtrent het laatste SQL-statement staat in cursor **SQL**

- Cursor is een pointer naar een stukje geheugen
- Expliciete cursor: aangemaakt door gebruiker (wordt niet behandeld!)
- Impliciete cursor: aangemaakt door de Oracle Server o.a. **SQL**
- Toegang tot de info in de cursor kan via cursor attributen:

<b>SQL%FOUND</b>	<b>Boolean attribute that evaluates to TRUE if the most recent SQL statement returned at least one row.</b>
<b>SQL%NOTFOUND</b>	<b>Boolean attribute that evaluates to TRUE if the most recent SQL statement did not return even one row.</b>
<b>SQL%ROWCOUNT</b>	<b>An integer value that represents number of rows affected by the most recent SQL statement.</b>

# Procedure oproepen

- Vanuit een andere procedure:

in het BEGIN-blok:

```
ADD_CTRY;
```

- Vanuit SQL\*Plus:

```
SQL> execute add ctry  
OF exec add_cTry
```

# Oefening 1

# Oefening 2

# Voorbeeld: procedure met IN parameter(s)

```
CREATE OR REPLACE PROCEDURE del_ctype
    (p_country_id      IN      countries.country_id%type)
IS
BEGIN
    DELETE FROM countries
    WHERE country_id = p_country_id;

    DBMS_OUTPUT.PUT_LINE('Er werden ' || SQL%ROWCOUNT ||
                          'rijen verwijderd uit de tabel COUNTRIES');

END del_ctype;
/
```



# PL/SQL Syntax - parameters

- Parameterlijst: zie hoofdstuk Functies
- IN-parameter komt binnen in de procedure (called program, subprogramma) en wordt meegegeven vanuit een andere procedure (calling program) of vanuit een aanroep in SQL\*Plus
- De parameter(s) in het called program worden FORMAL-parameters genoemd
- De parameter(s) in het calling program worden ACTUAL-parameters genoemd
- Sql> `DESC[RIBE] del_ctype` geeft informatie over de parameters van de procedure del\_ctype

# Procedure oproepen

- Vanuit een andere procedure:

in het BEGIN-blok:

```
del_ctry(v_country_id) ;
```

```
OF del_ctry('AR')
```

- Vanuit SQL\*Plus:

```
SQL> exec del_ctry('AR')
```

```
OF exec del_ctry('&landid')
```

# Oefening 3

# Voorbeeld: procedure met IN en OUT parameter(s)

```
CREATE OR REPLACE PROCEDURE raise_salary_dept
    (p_dept_name      IN      departments.department_name%type
    ,p_percent         IN      number
    ,p_count_emp       OUT     number)
AS
    v_dept_id departments.department_id%type;
BEGIN
    SELECT department_id INTO v_dept_id
    FROM departments
    WHERE department_name = p_dept_name;
    UPDATE employees
    SET salary = salary * (1 + p_percent/100)
    WHERE department_id = v_dept_id;
    p_count_emp := SQL%rowcount;
END raise_salary_dept;
```

## Voorbeeld: Andere mogelijke oplossing

```
CREATE OR REPLACE PROCEDURE raise_salary_2_dept
    (p_dept_name    IN        departments.department_name%type
    ,p_percent       IN        number
    ,p_count_emp     OUT       number)
AS
BEGIN
    UPDATE employees
    SET salary = salary * (1 + p_percent/100)
    WHERE department_id = (SELECT department_id FROM departments
                           WHERE department_name = p_dept_name);

    p_count_emp := SQL%rowcount;
END raise_salary_2_dept;
/
```

# Procedure met IN-en OUT parameter(s) oproepen

Een procedure kan worden opgeroepen

- vanuit een andere procedure
- via een anoniem block
- aan de SQL-prompt → bind-variable nodig

# Oproeping vanuit een andere procedure

.....

AS

    v\_aantal\_emp number(3);

BEGIN

    raise\_salary\_dept('Administration',10,v\_aantal\_emp);

    DBMS\_OUTPUT.PUT\_LINE(v\_aantal\_emp);

END;

- Vanuit deze procedure wordt de naam van het department nl. Administration en het percentage nl. 10 meegegeven aan het called program nl. raise\_salary\_dept
- Na het uitvoeren zal het aantal employees in het department Administration met een loonsverhoging van 10% worden afgedrukt

# Oproeping via een anoniem block

```
DECLARE
    v_aantal_emp    number(3);
BEGIN
    raise_salary_dept('Administration',10,v_aantal_emp);
    DBMS_OUTPUT.PUT_LINE(v_aantal_emp);
END;
/
```



# Oproeping via de SQL-prompt → bind-variable

- Bind variable
  1. deze variabele wordt gecreëerd in de werkomgeving en kan gebruikt worden in SQL statements en PL/SQL blocks
  2. syntax aan SQL-prompt: VARIABLE b\_test varchar(2)
  3. gebruikt als volgt :b\_test
- SQL> VARIABLE b\_aantal\_emp number  
SQL> exec raise\_salary\_dept('Administration', 10, :b\_aantal\_emp)
- Om afdruk van bind variable te zien voeg je de volgende setting toe in login.sql: SET AUTOPRINT ON

```
SQL> exec raise_salary_dept('Administration',10,:b_aantal_emp)
PL/SQL procedure successfully completed.

B_AANTAL_EMP          1
```

# Oefening

Probeer de procedure `raise_salary_2_dept` uit te voeren en te gebruiken ([Voorbeeld: Andere mogelijke oplossing](#))

# Positional vs. Named notation

- Voorbeeld positional notation  
`exec raise_salary_dept('Administration', 10, :b_aantal_emp)`

Volgorde van parameters moet exact dezelfde zijn als in de procedure

- Voorbeeld named notation  
`exec raise_salary_dept(p_percent =>10,p_dept_name  
=>'Administration',p_count_emp =>:b_aantal_emp)`

Volgorde is niet langer belangrijk maar de parameter-namen moeten dan wel gekend zijn

# Oefening 4

# Voorbeeld: procedure met eenvoudige LOOP

```
CREATE OR REPLACE PROCEDURE print_dept_loop
AS
    v_count          number(3) := 0;
    v_dept_id         departments.department_id%type;
    v_dept_name       departments.department_name%type;
    v_man_id          departments.manager_id%type;
BEGIN
    LOOP
        v_count := v_count + 10;
        SELECT department_id, department_name, manager_id
        INTO v_dept_id, v_dept_name, v_man_id
        FROM departments
        WHERE department_id = v_count;
        DBMS_OUTPUT.PUT_LINE(v_dept_id || ' ' || v_dept_name || ' ' || v_man_id);
        EXIT WHEN v_count >= 100;
    END LOOP;
END print_dept_loop;
/
```

**(herhaal tot ...)**

# Voorbeeld: procedure met WHILE LOOP

```
CREATE OR REPLACE PROCEDURE print_dept_while  
AS
```

```
    v_count          number(3) := 10;  
    v_dept_id        departments.department_id%type;  
    v_dept_name      departments.department_name%type;  
    v_man_id         departments.manager_id%type;
```

```
BEGIN
```

```
    WHILE v_count <= 100 LOOP  
        SELECT department_id, department_name, manager_id  
        INTO v_dept_id, v_dept_name, v_man_id  
        FROM departments  
        WHERE department_id = v_count;  
        DBMS_OUTPUT.PUT_LINE(v_dept_id||'   '||v_dept_name||'   '||v_man_id);  
        v_count := v_count + 10;  
    END LOOP;
```

```
END print_dept_while;
```

(zolang ...)

BIG DATA - PL/SQL - H3 - Procedures

# Voorbeeld: procedure met FOR LOOP

```
CREATE OR REPLACE PROCEDURE print_dept_for  
AS
```

```
    v_dept_id      departments.department_id%type;  
    v_dept_name    departments.department_name%type;  
    v_man_id       departments.manager_id%type;
```

```
BEGIN
```

```
    FOR i IN 1..10 LOOP
```

```
        SELECT department_id, department_name, manager_id
```

```
        INTO v_dept_id, v_dept_name, v_man_id
```

```
        FROM departments
```

```
        WHERE department_id = i*10;
```

```
        DBMS_OUTPUT.PUT_LINE(v_dept_id||'   '||v_dept_name||'   '||v_man_id);
```

```
    END LOOP;
```

```
END print_dept_for;
```

```
/
```

**(zelftellende lus)**

**OPM : teller i wordt impliciet gedeclareerd en kan enkel in lus worden gebruikt – kan wel toegewezen worden aan een variabele en deze is bruikbaar buiten de lus**

# Oefening 5

# Oefening 6



# Speciale variant van de FOR-loop – Cursor Loop

```
CREATE OR REPLACE PROCEDURE print_dept_cursorloop
AS
BEGIN
    FOR rec IN (SELECT department_id, department_name, manager_id
                FROM departments
                WHERE department_id between 10 and 100)
    LOOP
        DBMS_OUTPUT.PUT_LINE(rec.department_id ||
                              '||rec.department_name||'
                              '||rec.manager_id);
    END LOOP;
END print_dept_cursorloop;
```

# Speciale variant van de FOR-loop – Cursor Loop

- Alle rijen en kolommen bekomen door het uitvoeren van de subquery worden in een expliciete cursor bijgehouden.
- Deze cursor heeft geen naam en daarom kan er ook geen gebruik gemaakt worden van cursorattributen  
=> `rec%rowcount` kan NIET GEBRUIKT worden
- De FOR-loop zal rij per rij verwerken
- In de LOOP kan er verwezen worden naar een specifiek attribuut via `rec.department_name` – het gaat hier dan over de inhoud van het attribuut `department_name` in de rij die op dat moment door de loop verwerkt wordt

# Oefening 7

# Oefening 8

# Procedures verwijderen

- Syntax:

```
DROP PROCEDURE procedure_name
```

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

# Opvragen kenmerken (data dictionary)

Alle informatie over bestaande PL/SQL procedures 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, procedures,....
- **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 procedures aanwezig zijn:

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

# 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 = 'raise_salary_dept';
```

# Oefening 9

# Oefening 10



# PL/SQL H4

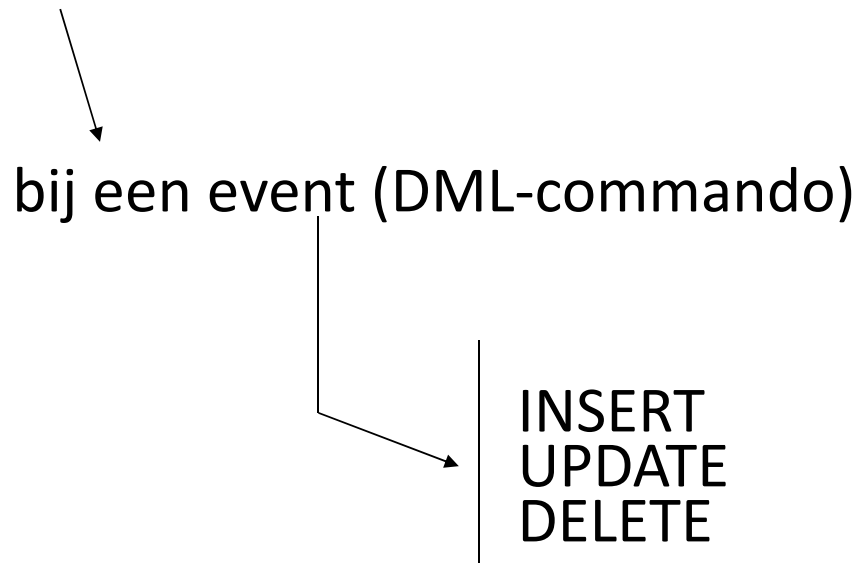
## Database Triggers

HOGESCHOOL 



# Inleiding Database Triggers

- PL/SQL-code geassocieerd aan een tabel/view
- wordt als object opgeslagen in de DB
- wordt automatisch uitgevoerd (kan niet opgeroepen worden)



# Inleiding Database Triggers

- inhoud van de tabel wijzigt (insert/update/delete)  
 trigger wordt automatisch uitgevoerd

!!!!!! het maakt niet uit vanuit welke omgeving of door wie de gegevens worden gewijzigd!!!!!!

# Syntax voor de creatie van een database trigger

CREATE OR REPLACE TRIGGER *triggernaam*

{BEFORE / AFTER}

→ ***timing***

{DELETE / INSERT / UPDATE [OF *kolom* [,*kolom*] ] } [ OR...]

→ ***event***

ON *tabelnaam*

[FOR EACH ROW [WHEN (*voorwaarde*) ] ]

→ ***frequency***

[DECLARE ]

BEGIN

*/\*uitvoerbare commando's\*/*

[EXCEPTION]

END [*triggernaam*];

# Syntax tabel triggers


- **timing**
  - BEFORE the event
  - AFTER the event
  - ☐ BEFORE
    - als de trigger moet controleren of een actie toegestaan is of niet - voorkomt onnodige rollbacks
    - als je zeker wil zijn dat deze trigger altijd afgaat (deze worden eerst uitgevoerd)
  - ☐ AFTER
    - als het triggerende commando zeker moet worden uitgevoerd (vooral bij row triggers)
    - wordt pas uitgevoerd nadat alle constraints op de tabel gecontroleerd zijn
- **event**
  - INSERT – UPDATE - DELETE
- **frequency**
  - STATEMENT (default)
  - ROW
  - = aantal keer trigger-body uitgevoerd wordt
  - ☐ statement:
    - 1x per statement / instructie
    - onafhankelijk van het aantal beïnvloede rijen
  - ☐ row:
    - 1x per rij die beïnvloed wordt door het event

# Voorbeeld: Statement Trigger

```
CREATE OR REPLACE TRIGGER bds_emp
  BEFORE DELETE
  ON employees
BEGIN
  IF USER != 'JAN' THEN
    RAISE_APPLICATION_ERROR(-20000,
      'u heeft geen rechten voor deze actie');
  END IF;
END;
/
```

→ creatie als object in de databank

# Foutmelding via RAISE\_APPLICATION\_ERROR

- RAISE\_APPLICATION\_ERROR(-20000, 'u heeft geen rechten voor deze actie')  


foutcode      foutmelding(moet string zijn)
- Een SQL-commando waarmee een fout gecreëerd/geraised wordt en op het scherm wordt afgedrukt (bruikbaar in elke applicatie)
  - het programma/trigger wordt afgebroken
  - automatisch ROLLBACK voor het triggerende DML-statement(hier delete)
- Foutcode moet liggen tussen -20000 en -20999 (user-defined)

# Naamgeving triggers

De naamgeving geeft het soort trigger weer

**vb: TRIGGER bds\_emp**

- b = before  
de trigger gaat af vóór het DML-statement wordt uitgevoerd
- d = delete  
de trigger gaat af bij een delete-statement
- s = statement trigger  
deze trigger gaat slechts 1x af per DML-statement, ongeacht hoeveel rijen door het DML-commando worden bewerkt



## / → creatie trigger

- Via R(un) of / indien code in buffer of via start [naam sql-bestand]
- de broncode wordt in ieder geval 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`

# Trigger gebruiken?

- De tabel trigger gaat automatisch af bij het uitvoeren van een DML-statement op een specifieke tabel waarvoor een trigger gecompileerd is
- Er kunnen meerdere triggers op 1 tabel gecreëerd worden (volgorde van uitvoeren: zie later)

# Oefening 1

# Voorbeeld: Statement Trigger uitgebreid

```
CREATE OR REPLACE TRIGGER bdus_emp
  before delete or update of salary
  ON employees
BEGIN
  IF USER != 'JAN' THEN
    IF DELETING THEN
      RAISE_APPLICATION_ERROR(-20000, 'u heeft geen verwijderrechten');
    ELSE
      RAISE_APPLICATION_ERROR(-20000, 'u heeft geen rechten om het salary te wijzigen');
    END IF;
  END IF;
END;
/
```

# Functies INSERTING – DELETING - UPDATING

- Te gebruiken indien er meer dan 1 triggerend event is op 1 tabel
- Deze functies geven een boolean terug
- Bij UPDATING kan ook een parameter meegegeven worden  
vb. IF updating('salary') THEN .....

# Oefening 2

# Oefening 3

# Voorbeeld: Row Trigger

```
CREATE OR REPLACE TRIGGER aur_emp_salary
  AFTER UPDATE OF salary
  ON employees
  FOR EACH ROW
BEGIN
  IF (:NEW.salary - :OLD.salary > 0.1* :OLD.salary) THEN
    RAISE_APPLICATION_ERROR(-20000, 'salary te veel
                                   verhoogd' );
  END IF;
END;
/
```

# Row triggers

- bovenaan de trigger definitie: FOR EACH ROW
- gaat per te bewerken rij af
  - Vb een delete-commando verwijdt 10 rijen
    - statement-trigger gaat **1 keer** af
    - rij-trigger gaat **10 keer** af
  - Vb een delete-commando verwijdt 0 rijen
    - statement-trigger gaat **1 keer** af
    - rij-trigger gaat **niet** af



# Row triggers

- mogelijk oude en nieuwe kolomwaarden op te vragen
  - :NEW.kolomnaam
  - :OLD.kolomnaam

vb. :NEW.salary      bevat de nieuwe waarde voor salary na uitvoering van een INSERT of UPDATE  
Let op: bij DELETE is deze variabele leeg

:OLD.salary      bevat de oude waarde voor salary vóór uitvoering van een UPDATE of DELETE  
Let op: bij INSERT is deze variabele leeg

Enkel bij UPDATE bevatten beiden een waarde

# Voorbeeld: Row Trigger uitbreiding (WHEN)

```
CREATE OR REPLACE TRIGGER aur_emp_sal2
  AFTER UPDATE OF salary
  ON employees
  FOR EACH ROW
  WHEN (OLD.job_id != 'AD_PRES')
BEGIN
  IF (:NEW.salary - :OLD.salary > 0.1* :OLD.salary) THEN
    RAISE_APPLICATION_ERROR(-20000, 'salary te veel
      verhoogd');
  END IF;
END;
/
```

Enkel mogelijk voor row triggers!

LET OP: geen ':' bij 'OLD'

# Oefening 4

## Volgorde van uitvoering triggers (automatisch)

Indien meerdere triggers op 1 DML-statement

1. Alle BEFORE STATEMENT triggers
2. Voor elke rij uit de ROW triggers
  - a. Alle BEFORE ROW triggers voor die rij
  - b. Triggerende DML-statement + integrity constraints checken voor die rij
  - c. Alle AFTER ROW triggers voor die rij
3. Alle AFTER STATEMENT triggers

# Trigger keuze

- Gebruik een **row trigger** als de inhoud van de kolommen nodig is
- Gebruik een **before statement trigger** als de trigger MOET afgaan
- Gebruik eerder een **before statement-trigger** dan een after statement trigger  
(vooral bij controle of een actie toegestaan is, dit voorkomt rollbacks)
- Gebruik liever een **after row trigger** dan een before row trigger  
Oracle controleert dan eerst de constraints.
- Gebruik een **before row trigger** als de inhoud van een kolom in de trigger gewijzigd wordt

# Beperkingen van Database Triggers

Commando's COMMIT en ROLLBACK zijn niet toegelaten in triggers

# Oefening 5

# Oefening 6

# Beheer van triggers

- **Welke triggers bestaan?**

```
SQL>SELECT object_name, created, status  
      FROM user_objects  
      WHERE object_type = 'TRIGGER';
```

- **Broncode opvragen:**

```
SQL> SELECT line, text  
      FROM user_source  
      WHERE name = 'AUR_EMP_SALARY';
```



## Beheer van triggers – tabel **USER\_TRIGGERS**

- SQL> SELECT trigger\_type, trigger\_body  
FROM **user\_triggers**  
WHERE trigger\_name = 'AUR\_EMP\_SALARY';
- SQL> SELECT trigger\_name, trigger\_type,  
triggering\_event, table\_name, status  
FROM **user\_triggers**;

# Beheer van triggers

- **Verwijderen**

DROP TRIGGER triggernaam

Alle privileges betreffende de trigger worden mee verwijderd.

- De CREATE OR REPLACE syntax is equivalent aan het verwijderen en opnieuw creëren van de trigger. Toegekende privileges i.v.m. de trigger blijven bestaan als deze syntax gebruikt wordt.

- **Activeren/deactiveren van 1 bepaalde trigger**

ALTER TRIGGER triggernaam ENABLE

ALTER TRIGGER triggernaam DISABLE

- **Activeren/deactiveren van alle triggers van 1 bepaalde tabel**

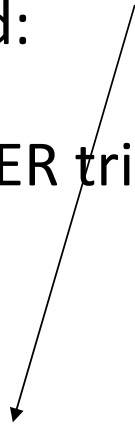
ALTER TABLE tabelnaam ENABLE ALL TRIGGERS

ALTER TABLE tabelnaam DISABLE ALL TRIGGERS

# Beheer van triggers

- een trigger die **invalid** geworden is, kan opnieuw worden gecompileerd:

```
ALTER TRIGGER triggernaam COMPILE
```



*Hoe kan een trigger invalid worden?*

wanneer bvb een wijziging van de structuur van de gerefereerde tabel optreedt

# Oefening 7

## Precedentie

(...), *, /, +, -,   , < > =, IS LIKE IN, BETWEEN, !=, NOT, AND, OR	volgorde van bewerkingen
---	--------------------------

## Vergelijkingsoperatoren

<, >, <=, >=	kleiner dan, groter dan, ... of gelijk aan
!=   <>   ^=	verschillend van
=	is gelijk aan
IS [NOT] NULL	is (niet) gelijk aan null
BETWEEN ondergrens AND bovengrens	tussen twee waarden, inclusief de twee waarden
IN (waarde1, waarde2,...)	is gelijk aan een van de waarden tussen de haakjes
LIKE 'patroon'	komt overeen met een patroon, hoofdlettergevoelig
%	willekeurig aantal karakters
_	één willekeurig teken
'patroon' ESCAPE 'c'	karakter c voor letterlijke tekens

## Logische operatoren

AND	vereist dat beide voorwaarden waar zijn
NOT	vereist dat de voorwaarde niet waar is (o.a. NOT IN, NOT BETWEEN, ...)
OR	vereist dat minstens één voorwaarde waar is

## Groeperende functies

AVG(expr)	Gemiddelde (NULL waarden tellen niet mee)
COUNT(expr)	aantal rijen (NULL waarden tellen niet mee)
MAX(expr)	maximum waarde
MIN(expr)	minimum waarde
SUM(n)	som

## Conditionele functies

CASE WHEN c1 THEN r1 ... ELSE rn END	Als c1 waar is dan r1... en anders rn
COALESCE(a, b, c, ...)	het eerste niet-null argument
DECODE(x, s1, r1,s2, r2,...default)	r1 als x=s1, r2 als x=s2, ...en anders default
NULLIF(expr1, expr2)	Null als expr1=expr2 en anders expr1
NVL(expr1, expr2)	expr2 als expr1 NULL is, anders expr1
NVL2(expr1, expr2, expr3)	expr2 als expr1 niet NULL is, en anders expr3

## Conversiefuncties

TO_CHAR(expr, fmt)	conversie van expr naar een tekst
TO_DATE(char, fmt)	conversie van char naar een datum
TO_NUMBER(char, fmt)	conversie van char naar een getal

## Formaatmodel

"tekst"	tekst, letterlijk in het formaat
Day	dag van de week voluit
DD	numerieke dag van de maand
DY	dag van de week in drie letters
Fm	fill mode: verwijdert opvulspaties of onderdrukt beginnullen
HH of HH12 of HH24	Uur van de dag, of uur (1-12), of uur (0-23)
MI	minuten (0-59)
MON	maand, drie letters
Month	maand, voluit geschreven
MM	maand, in 2 cijfers
RR	jaar in 2 cijfers
SCC of CC	eeuw
SP en SPTH	uitgespeld getal, bijv. 'vier' en 'vierde'
SS	seconden (0-59)
SSSSS	seconden na middernacht
TH en THSP	ordinaal getal, bijv. '4e' en 'vierde'
Year	jaar uitgespeld
YYYY	jaar in 4 cijfers

## Getallen

* + - /	rekenkundige expressies
MOD(deler, deeltal)	modulo, de rest van deler na deling door deeltal
ROUND(getal[, n])	rondt getal af op n cijfers na de komma (geen n is 0, n < 0 is voor de komma).
TRUNC(getal[, n])	kapt getal af op n cijfers na de komma (geen n is 0, n < 0 is voor de komma).

## Tekst

'There''s Oracle SQL'	Tekst, hoofdlettergevoelig, 2 keer ' voor een aanhalingsteken
	concatenatie
CONCAT(c1, c2)	voegt de tekst c1 en c2 samen
INITCAP(tekst)	zet de eerste letter van elk woord in hoofdletters, de rest in kleine letters
INSTR(c1, c2[, start[, n]])	vindt de positie van het n <sup>de</sup> voorkomen van c2 in c1, vanaf positie start. Het eerste karakter heeft positie 1. Negatieve argumenten tellen vanaf het eind.
LENGTH(tekst)	geeft aantal karakters terug
LOWER(tekst)	zet tekst om naar kleine letters
LPAD(c1, n, c2)	vult c1 wordt links aan tot lengte n met karakters van c2
REPLACE(tekst, s, r)	leder voorkomen van s in tekst vervangen door r
RPAD(c1, n, c2)	vult c1 rechts aan tot lengte n met karakters van c2
SUBSTR(tekst, start[, n])	tekstdeel uit tekst vanaf positie start, n karakters lang (of tot het einde). Het eerste karakter heeft positie 1. Negatieve argumenten tellen vanaf het eind.
TRIM([c1 FROM] c2)	verwijdert karakters c1 of spaties aan het begin en eind van c2
UPPER(tekst)	zet tekst om naar hoofdletters

## Datums

datum + n	telt n dagen op bij datum (decimaal getal voor uren, minuten, ...)
datum1 – datum2	geeft aantal dagen (getal) tussen twee datums (kommagetal bij uren, ...)
ADD_MONTHS(datum, n)	telt n maanden op bij datum (n mag negatief zijn)
CURRENT_DATE	datum en tijd in de tijdzone van de huidige sessie (lokaal)
LAST_DAY(datum)	de laatste dag van de maand waarin datum valt
MONTHS_BETWEEN(dt1, dt2)	aantal maanden tussen twee datums, positief als dt1 na dt2 valt (cijfers na de komma stellen een deel van de maand voor).
NEXT_DAY(datum, 'dag')	de eerste weekdag 'dag' (tekst of getal) gelijk aan of later dan datum
ROUND(datum[, 'fmt'])	rondt datum af tot dichtstbijzijnde eenheid fmt (of 'dag')
SYSDATE	datum en tijd van de database
TRUNC(datum[, 'fmt'])	kapt datum af op eenheid fmt (of dag)

## Substitutievariabelen

&var	variabele var, letterlijk in te voegen in de SQL de waarde wordt gevraagd aan de gebruiker indien nodig
'&var'	variabele var, gebruikt als tekst (hoofdlettergevoelig) of datum
&&var	variabele var, onthoud de waarde voor hergebruik
DEFINE var = value	variabele var de waarde value toewijzen, onthoud de waarde
SET VERIFY [ON OFF]	toon wel/niet de SQL met de variabelen ingevuld
UNDEFINE var	variabele var vergeten

## Algemeen

*	alle kolommen
AS "Kolom naam"	kolom alias, AS is optioneel
DESCRIBE	weergave tabelstructuur
DISTINCT	onderdrukt dubbels
DUAL	ingebouwde dummy tabel
FROM tabel1 t1	gebruik data uit tabel1, met alias t1
GROUP BY	waarop moeten rijen gegroepeerd worden
HAVING voorwaarde	filtert de rijen na groepering met de voorwaarde
tabel1 t1 JOIN tabel2 t2 USING veldnaam	
tabel1 t1 JOIN tabel2 t2 ON voorwaarde	
...LEFT OUTER JOIN...	alle velden linkertabel worden getoond
...RIGHT OUTER JOIN...	alle velden rechtertabel worden getoond
...FULL OUTER JOIN...	alle velden van beide tabellen worden getoond
...CROSS JOIN...	product van twee tabellen
tabel1 t1, tabel2 t2	product van twee tabellen
ORDER BY criteria1, criteria2, ...	sorteert (oplopend) op basis van kolom, kolomalias, kolompositie of een expressie.
criteria [ASC DESC] [NULLS FIRST LAST]	af- of oplopend, met lege waarden eerst of laatst.
SELECT t1.kolom	selectie uit tabel(len)
SHOW ALL	toon alle systeemvariabelen.
WHERE voorwaarde	filtert de rijen met de voorwaarde