

# PL/SQL H4

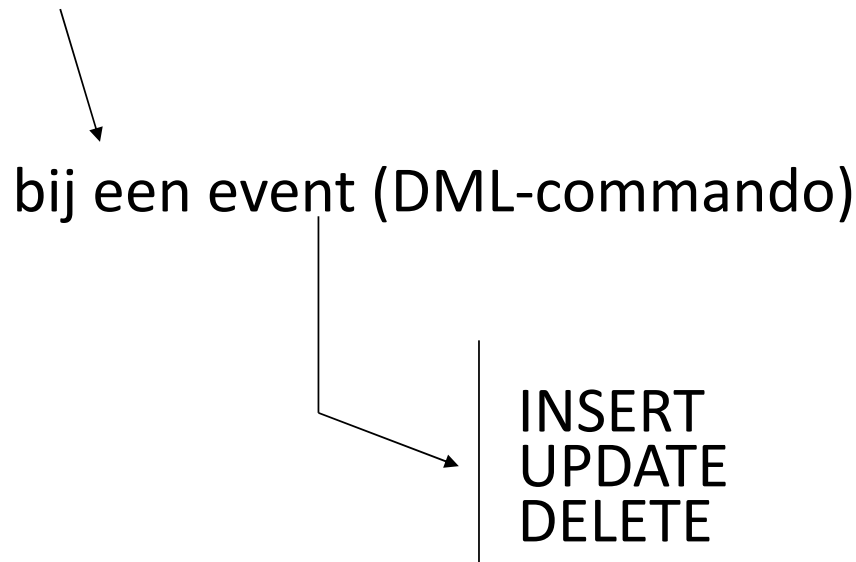
## Database Triggers

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

Oefening 5

## 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 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

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

ALTER TRIGGER triggernaam ENABLE

ALTER TRIGGER triggernaam DISABLE

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

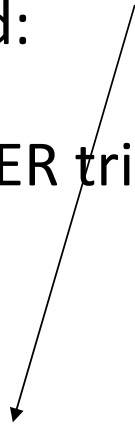
ALTER TABLE tabelnaam ENABLE ALL TRIGGERS

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