Trigger

Trigger-Arten

Trigger können in PL/SQL für folgende Ereignisse (Auslöser) verwendet werden:

- DML-Anweisungen (`DELETE`, `INSERT`, `UPDATE`)
- DDL-Anweisungen (`CREATE`, `ALTER`, `DROP`)
- Datenbank-Operationen wie `SERVERERROR`, `LOGON`, `LOGOFF`, `STARTUP`, `SHUTDOWN`

Anwendungsfälle für Trigger

- Sicherheit
- Auditing
- Datenintegrität
- Replikation von Tabellen
- Berechnung abgeleiteter Daten
- Protokollierung

DML-Trigger

- Der Ereignis-Typ bestimmt welche DML-Anweisung die Auslösung des Triggers verursacht. Die möglichen Ereignisse sind:
  - `INSERT`
  - `UPDATE [OF column]`
  - `DELETE`
- Der Trigger-Body bestimmt, welche Aktionen ausgeführt werden und besteht aus einem PL/SQL-Block oder einem Aufruf an eine Prozedur.

DML-Trigger: CREATE TRIGGER Anweisung

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing -- when to fire the trigger
event1 [OR event2 OR event3]
ON object_name
[REFERENCING OLD AS old | NEW AS new]
FOR EACH ROW -- default is statement level trigger
WHEN (condition)]]
DECLARE]
BEGIN
... trigger_body -- executable statements
[EXCEPTION . . .]
END [trigger_name];
```

```
timing =  BEFORE | AFTER | INSTEAD OF
```

```
event = INSERT | DELETE | UPDATE | UPDATE OF column_list
```

Wann wird der Trigger gefeuert

- `BEFORE`: Feuert den Trigger bevor die DML-Anweisung auf die Tabelle angewendet wird.
- `AFTER`: Feuert den Trigger nachdem die DML-Anweisung auf die Tabelle angewendet wurde.
- `INSTEAD OF`: Feuert den Trigger statt die DML-Anweisung auszuführen. Wird typischerweise für VIews benutzt, die ansonsten nicht schreibbar sind.

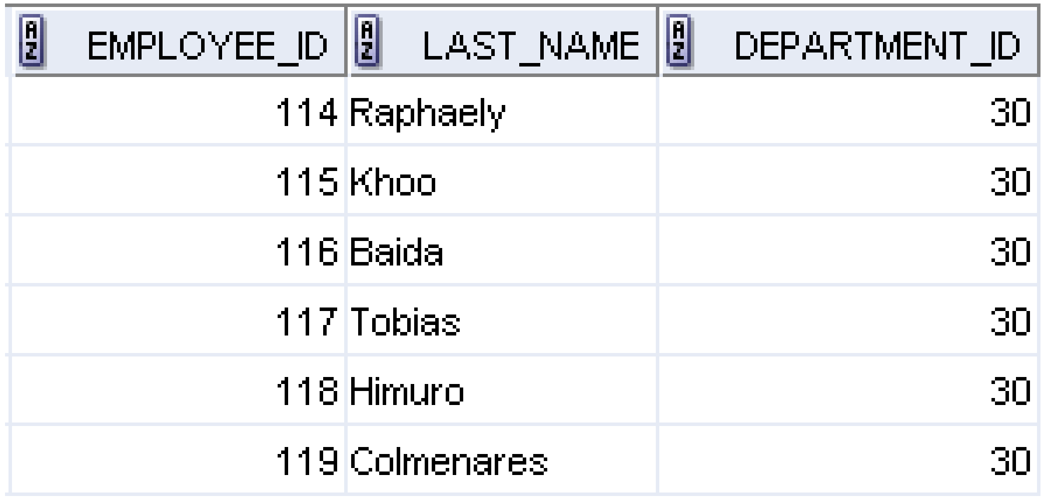
Statement-Level Trigger Versus Row-Level Trigger

- Statement-Level Trigger:
  - Default bei Erzeugung eines Triggers
  - Wird einmal je DML-Anweisung gefeuert, auch wenn keine Zeile getroffen wird
- Row-Level Trigger
  - FOR EACH ROW Klausel muss verwendet werden
  - Wird je getroffener Zeile gefeuert
  - Falls keine Zeile getroffen wird, kommt auch der Trigger nicht zur Ausführung

Ablaufsequenz

```
UPDATE employees
  SET salary = salary * 1.1
  WHERE department_id = 30;
```

BEFORE statement trigger



EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
114	Raphaely	30
115	Khoo	30
116	Baida	30
117	Tobias	30
118	Himuro	30
119	Colmenares	30

BEFORE row trigger

AFTER row trigger

. . .

BEFORE row trigger

AFTER row trigger

. . .

AFTER statement trigger

DML-Trigger Beispiel

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT ON employees
  BEGIN
   IF (TO_CHAR(SYSDATE,'DY') IN
('SAT','SUN')) OR
      (TO_CHAR(SYSDATE,'HH24:MI')
          NOT BETWEEN '08:00' AND '18:00')
THEN
   RAISE_APPLICATION_ERROR(-20500, 'You may
insert'
      ||' into EMPLOYEES table only during '
      ||' normal business hours.');
   END IF;
  END;
```

Testen des Triggers

```
INSERT INTO employees (employee_id, last_name,
    first_name, email, hire_date,
job_id, salary, department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE,
  'IT_PROG', 4500, 60);
```

Detailsteuerung über Prädikate

```
CREATE OR REPLACE TRIGGER secure_emp BEFORE
INSERT OR UPDATE OR DELETE ON employees
  BEGIN
    IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
       (TO_CHAR(SYSDATE,'HH24')
        NOT BETWEEN '08' AND '18') THEN
      IF DELETING THEN RAISE_APPLICATION_ERROR(
        -20502,'You may delete from EMPLOYEES table'||
        'only during normal business hours.');
      ELSIF INSERTING  THEN RAISE_APPLICATION_ERROR(
        -20500,'You may insert into EMPLOYEES table'||
        'only during normal business hours.');
      ELSIF UPDATING  ('SALARY') THEN
        RAISE_APPLICATION_ERROR(-20503, 'You may '||
        'update SALARY only normal during business hours.');
      ELSE RAISE_APPLICATION_ERROR(-20504,'You may'||
        ' update EMPLOYEES table only during'||
        ' normal business hours.');
      END IF;
    END IF;
  END;
```

DML Row Trigger Beispiel

```
CREATE OR REPLACE TRIGGER restrict_salary
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
  IF NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP'))
     AND :NEW.salary > 15000 THEN
    RAISE_APPLICATION_ERROR (-20202,
      'Employee cannot earn more than $15,000.');
  END IF;
END;/
```

```
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell';
```

Zugriff auf die Zielwerte

- Für Row-Level-Trigger werden 2 Datenstrukturen zur Verfügung gestellt:
  - `OLD`: Speichert die ursprünglichen Werte der Zeile, die durch die DML-Anweisung getroffen wird.
  - `NEW`: Enthält die neuen Werte
- `NEW` und `OLD` haben Record-Struktur gemäß `%ROWTYPE` für die Zieltabelle

Data Operations	Old Value	New Value
`INSERT`	`NULL`	Inserted value
`UPDATE`	Value before update	Value after update
`DELETE`	Value before delete	`NULL`

Beispiel für die Nutzung

```
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO audit_emp(user_name, time_stamp, id,
    old_last_name, new_last_name, old_title,
    new_title, old_salary, new_salary)
  VALUES (USER, SYSDATE, :OLD.employee_id,
    :OLD.last_name, :NEW.last_name, :OLD.job_id,
    :NEW.job_id, :OLD.salary, :NEW.salary);
END;
/
```

Bedingtes Auslösen eines Row Triggers

```
CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
WHEN (NEW.job_id = 'SA_REP')
BEGIN
 IF INSERTING THEN
   :NEW.commission_pct := 0;
 ELSIF :OLD.commission_pct IS NULL THEN
   :NEW.commission_pct := 0;
 ELSE
   :NEW.commission_pct := :OLD.commission_pct+0.05;
 END IF;
END;
/
```

Implementierung einer Integritätsregel durch einen After Trigger

```
-- Integrity constraint violation error -2992 raised.
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
```

```
CREATE OR REPLACE TRIGGER employee_dept_fk_trg
AFTER UPDATE OF department_id
    ON employees FOR EACH ROW
BEGIN
 INSERT INTO departments VALUES(:new.department_id,
              'Dept '||:new.department_id, NULL, NULL);
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
    NULL; -- mask exception if department exists
END; /
```

```
-- Successful after trigger is fired
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
```

Erzeugung eines `INSTEAD OF` Trigger für DML-Operationen auf einem komplexen View

```
CREATE TABLE new_emps AS
 SELECT employee_id,last_name,salary,department_id
    FROM employees;

CREATE TABLE new_depts AS
 SELECT d.department_id,d.department_name,
        sum(e.salary) dept_sal
    FROM employees e, departments d
 WHERE e.department_id = d.department_id;

CREATE VIEW emp_details AS
 SELECT e.employee_id, e.last_name, e.salary,
        e.department_id, d.department_name
 FROM employees e, departments d
 WHERE e.department_id = d.department_id
GROUP BY d.department_id,d.department_name;
```

Erzeugung eines `INSTEAD OF` Trigger für DML-Operationen auf einem komplexen View

```
CREATE OR REPLACE TRIGGER new_emp_dept
INSTEAD OF INSERT OR UPDATE OR DELETE ON emp_details
FOR EACH ROW
BEGIN
  IF INSERTING THEN
    INSERT INTO new_emps
    VALUES (:NEW.employee_id, :NEW.last_name,
                 :NEW.salary, :NEW.department_id);
    UPDATE new_depts
      SET dept_sal = dept_sal + :NEW.salary
      WHERE department_id = :NEW.department_id;
  ELSIF DELETING THEN
    -- ...;
  END IF;
END;
```

Deaktivierter Trigger

- Ab Oracle Database 11g kann man einen deaktivierten Trigger erzeugen. Dieser kann später aktiviert werden.

```
CREATE OR REPLACE TRIGGER mytrg
  BEFORE INSERT ON mytable FOR EACH ROW
  DISABLE    -- Disabling Trigger
BEGIN
  :New.ID := my_seq.Nextval;
. . .
END;
```

ALTER und DROP bei Triggern

```
-- Disable or reenable a database trigger:

ALTER TRIGGER trigger_name DISABLE | ENABLE;
```

```
-- Disable or reenable all triggers for a table:

ALTER TABLE table_name DISABLE | ENABLE ALL TRIGGERS;
```

```
-- Recompile a trigger for a table:

ALTER TRIGGER trigger_name COMPILE;
```

```
-- Remove a trigger from the database:

DROP TRIGGER trigger_name;
```