Spezielle Trigger

Compound Trigger

- Ein DML-Trigger, der potentiell an mehreren Zeitpunkten ausgelöst wird:
  - Vor der auslösenden Anweisung (before)
  - Vor der Bearbeitung jeder getroffenen Zeile (before each row)
  - Nach der Bearbeitung jeder getroffenen Zeile (after each row)
  - Nach der auslösenden Anweisung (after)
- Zu jedem Zeitpunkt können unterschiedliche Aktionen ausgeführt werden
- Der Trigger besitzt Zustandsdaten, die von den Codierungen für die einzelnen Zeitpunkte gemeinsam genutzt werden können:
  - Der Zustand wird beim Start der auslösenden Anweisung initialisiert und bei Beendigung wieder frei gegeben
  - Dadurch ist ein Datenaustausch zwischen den einzelnen Phasen möglich

Struktur eines Compound Triggers

```
CREATE OR REPLACE TRIGGER schema.trigger
FOR dml_event_clause ON schema.table
COMPOUND TRIGGER

  -- Initial section
     -- Declarations
     -- Subprograms

  -- Optional section
  BEFORE STATEMENT IS ...;

  -- Optional section
  AFTER STATEMENT IS ...;

  -- Optional section
  BEFORE EACH ROW IS ...;

  -- Optional section
  AFTER EACH ROW IS ...;
```

Struktur eines Compound Triggers für eine Sicht (View)

```
CREATE OR REPLACE TRIGGER schema.trigger
FOR dml_event_clause ON schema.view
COMPOUND TRIGGER

-- Initial section
   -- Declarations
   -- Subprograms

-- Optional section (exclusive)
INSTEAD OF EACH ROW IS
...;
```

Restriktionen für Compound Trigger

- Ist DML-Trigger und steht somit nur für Tabellen und Views zur Verfügung
- Die Implementierung muss in PL/SQL erfolgen.
- Eine Exception, die in einer Sektion auftritt. kann auch nur dort behandelt werden. Es ist keine Weiterleitung an eine andere Sektion möglich.
- `:OLD` und `:NEW` können nicht im Deklarationsteil und den `BEFORE STATEMENT` oder `AFTER STATEMENT` Sektionen verwendet werden.
- Nur in der `BEFORE EACH ROW` Sektion kann der Wert von `:NEW` verändert werden

Das „Mutating Table“ Problem

- „Mutating Table“ kann sein:
  - Eine Tabelle, die gerade durch eine DML-Anweisung verändert wird.
  - Eine Tabelle, die gerade durch den Effekt eines `DELETE CASCADE` Constraint verändert wird.
- Während der Veränderung kann Code der Session nicht parallel auf die Tabelle zugreifen, um inkonsistente Zustände zu vermeiden. Dies gilt auch für den Code von Triggern.
- Diese Einschränkung gilt für alle Row Level Trigger (`FOR EACH ROW`) ausser `INSTEAD OF` Trigger für Views.

Mutating Table: Beispiel - 1

Umzusetzende Anforderung:
Das neue Gehalt soll im Bereich der bisherigen Gehälter liegen

```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE INSERT OR UPDATE OF salary, job_id
  ON employees
  FOR EACH ROW
  WHEN (NEW.job_id <> 'AD_PRES')
DECLARE
  v_minsalary employees.salary%TYPE;
  v_maxsalary employees.salary%TYPE;
BEGIN
  SELECT MIN(salary), MAX(salary)
   INTO   v_minsalary, v_maxsalary
   FROM   employees
   WHERE job_id = :NEW.job_id;
  IF :NEW.salary < v_minsalary OR :NEW.salary > v_maxsalary
    THEN RAISE_APPLICATION_ERROR(-20505,'Out of range');
  END IF;
END;
```

Mutating Table: Beispiel - 2

```
UPDATE employees
SET salary = 3400
WHERE last_name = 'Stiles';
```

```
TRIGGER check_salary Compiled.

Error starting at line 1 in command:
UPDATE employees
 SET salary = 3400
 WHERE last_name = 'Stiles'
Error report:
SQL Error: ORA-04091: table ORA42.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "ORA42.CHECK_SALARY", line 5
ORA-04088: error during execution of trigger 'ORA42.CHECK_SALARY'
04091. 00000 -  "table %s.%s is mutating, trigger/function may not see it"
*Cause:    A trigger (or a user defined plsql function that is referenced in
           this statement) attempted to look at (or modify) a table that was
           in the middle of being modified by the statement which fired it.
*Action:   Rewrite the trigger (or function) so it does not read that table.
```

Mutating Table: Beispiel - 3
Lösung mittels Compound Trigger

```
CREATE OR REPLACE TRIGGER check_salary
  FOR INSERT OR UPDATE OF salary, job_id
  ON employees
  WHEN (NEW.job_id <> 'AD_PRES')
  COMPOUND TRIGGER

  TYPE salaries_t             IS TABLE OF employees.salary%TYPE;
  min_salaries                salaries_t;
  max_salaries                salaries_t;

  TYPE department_ids_t        IS TABLE OF employees.department_id
    %TYPE;
  department_ids               department_ids_t;

  TYPE department_salaries_t  IS TABLE OF employees.salary%TYPE
                                INDEX BY VARCHAR2(80);
  department_min_salaries     department_salaries_t;
  department_max_salaries     department_salaries_t;

  -- continues on next slide
```

Mutating Table: Beispiel - 4

```
BEFORE STATEMENT IS
  BEGIN
    SELECT MIN(salary), MAX(salary), NVL(department_id, -1)
     BULK COLLECT INTO  min_Salaries, max_salaries, department_ids
    FROM    employees
    GROUP BY department_id;
    FOR j IN 1..department_ids.COUNT() LOOP
      department_min_salaries(department_ids(j)):= min_salaries(j);
      department_max_salaries(department_ids(j)):= max_salaries(j);
    END LOOP;
END BEFORE STATEMENT;

AFTER EACH ROW IS
  BEGIN
    IF :NEW.salary < department_min_salaries(:NEW.department_id)
      OR :NEW.salary > department_max_salaries(:NEW.department_id)
    THEN RAISE_APPLICATION_ERROR(-20505,'out of acceptable range');
    END IF;
  END AFTER EACH ROW;
END check_salary;
```

DDL-Trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name
BEFORE | AFTER -- Timing
[ddl_event1 [OR ddl_event2 OR ...]]
ON {DATABASE | SCHEMA}
trigger_body
```

DDL Events	Ausgelöst durch
CREATE	Beliebiges Datenbankobjekt wird erzeugt
ALTER	Beliebiges Datenbankobjekt wird verändert.
DROP	Beliebiges Datenbanobjekt wird gelöscht.

DDL-Trigger: Beispiel

```
CREATE OR REPLACE TRIGGER drop_trigger
  BEFORE DROP ON hr.SCHEMA
  BEGIN
    RAISE_APPLICATION_ERROR (
      num => -20000,
      msg => 'Cannot drop object');
  END;
/
```

System-Trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name
BEFORE | AFTER -- timing
[database_event1 [OR database_event2 OR ...]]
ON {DATABASE | SCHEMA}
trigger_body
```

Database Event	Ausgelöst durch
AFTER SERVERERROR	Oracle Error
AFTER LOGON	Anmeldung eines Nutzers
BEFORE LOGOFF	Abmelden eines Nutzers
AFTER STARTUP	Öffnen der Datenbank
BEFORE SHUTDOWN	Runterfahren

System-Trigger: Beispiel

```
-- Create the log_trig_table shown in the notes page
-- first

CREATE OR REPLACE TRIGGER logon_trig
AFTER LOGON  ON  SCHEMA
BEGIN
  INSERT INTO log_trig_table(user_id,log_date,action)
  VALUES (USER, SYSDATE, 'Logging on');
END;
/
```

```
CREATE OR REPLACE TRIGGER logoff_trig
BEFORE LOGOFF  ON  SCHEMA
BEGIN
  INSERT INTO log_trig_table(user_id,log_date,action)
  VALUES (USER, SYSDATE, 'Logging off');
END;
/
```