# PL/SQL
# Kontrollstrukturen

# Bedingte Verzweigungen

Syntax:

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

Beispiel:
Gib eine verbale Bewertung aus, abhängig vom Wert der Note.

```
IF  v_grade <= 1.5 THEN
     dbms_output.put_line('sehr gut');
ELSIF v_grade<= 2.5 THEN
     dbms_output.put_line('gut');
...
ELSE
     dbms_output.put_line('Kein gültiger Wert der Note');
END IF;
```

# CASE-Anweisung (1)

Syntax:

Auswahl über Bedingungen

```
CASE
  WHEN condition1 THEN statements;
  WHEN condition2 THEN statements;
  ...
[ELSE statements]
END CASE;
```

Beispiel:

```
CASE
  WHEN v_grade = 1 THEN dbms_output.put_line('sehr gut');
  WHEN v_grade = 2 THEN dbms_output.put_line('gut');
  WHEN v_grade = ...;
ELSE dbms_output.put_line('Kein gültiger Wert der Note');
END CASE;
```

# CASE-Anweisung (2)

Syntax:

Auswahl über Literale

```
CASE selector
  WHEN literal1 THEN statements;
  WHEN literal2 THEN statements;
  ...
[ELSE statements]
END CASE;
```

Beispiel:

```
CASE v_grade
  WHEN 1 THEN dbms_output.put_line('sehr gut');
  WHEN 2 THEN dbms_output.put_line('gut');
  WHEN ...;
ELSE dbms_output.put_line('Kein gültiger Wert der Note');
END CASE;
```

# Schleifen

**Schleifenarten in PL/SQL:**

- LOOP-EXIT-Schleifen
- FOR-Schleifen
- WHILE-Schleifen

# LOOP-EXIT-Schleife

- Abbruch mit/ohne Bedingung an beliebiger Stelle innerhalb der Schleife
- Iteration solange bis EXIT-Punkt erreicht

Syntax:

```
LOOP
   statement1;
   . . .
   EXIT [WHEN condition];
END LOOP;
```

Beispiel:

```
DECLARE
  counter NUMBER(2) := 1;
BEGIN
  LOOP
    dbms_output.put_line(counter);
    counter := counter + 1;
    EXIT WHEN counter > 10;
  END LOOP;
END;
```

# FOR-Schleife

Syntax:

```
FOR counter IN [REVERSE] lower_bound..upper_bound LOOP
   statements;
END LOOP;
```

- Der Iterator wird dabei implizit deklariert.
- IN REVERSE dekrementiert den Index bei jedem Schleifendurchlauf.

Beispiel: Einfügen der ArtikelID für die ersten 10 Gegenstände der Bestellung 601

```
DECLARE
  v_ordid item.ordid%TYPE := 601;
BEGIN
  FOR i IN 1..10 LOOP
    INSERT INTO item(ordid, itemid)
    VALUES(v_ordid, i);
  END LOOP;
END;
```

# WHILE-Schleife

Syntax:

```
WHILE condition LOOP
   statements;
END LOOP;
```

- Überprüfung der Bedingung zu Beginn des Schleifendurchlaufs
- Iteration, solange die Bedingung erfüllt ist

Beispiel: Einfügen der ArtikelID für die ersten 10 Gegenstände der Bestellung 601

```
DECLARE
  v_ordid item.ordid%TYPE := 601;
  v_counter PLS_INTEGER := 1;
BEGIN
  WHILE v_counter < 10 LOOP
    INSERT INTO item(ordid, itemid)
    VALUES(v_ordid, v_counter);
    v_counter := v_counter + 1;
  END LOOP;
END;
```

# SQL Cursor

**Problem:**

```
DECLARE
   v_ename emp.ename%TYPE;
   v_sal   emp.sal%TYPE;

BEGIN

   SELECT ename, sal
   INTO v_ename, v_sal
   FROM emp
   WHERE empno=7839;
   -- funktioniert

   SELECT ename, sal
   INTO v_ename, v_sal
   FROM emp;
   --Rückgabe einer Menge von DS
   --Speicherung in Variable nicht möglich! → Fehlermeldung
   --Deklaration eines Cursors erforderlich

END;
```

# SQL Cursor

- **Cu**rrent **S**et **o**f **R**ecords
  temporärer Bereich im Arbeitsspeicher
- Struktur zur Traversierung von Datensätzen
  Analogie in höheren Programmiersprachen: Iterator
- Typische Verwendung:
  Satzweise Verarbeitung von Daten
- Definitionsarten
  Implizite Cursor: für jeden SELECT- und DML-Befehl
  Explizite Cursor: explizit im Programm deklariert

# Expliziter Cursor

Syntax:

```
CURSOR cursor_name IS
select_statement;
```

- keine INTO-Klausel in der Cursor-Deklaration
- Verarbeitung der Zeilen in bestimmter Reihenfolge erforderlich: ORDER BY

Beispiel:

```
DECLARE
CURSOR emp_cursor IS
  SELECT ename, sal
  FROM emp;
...
```

# Öffnen eines Cursors

Aktivitäten beim Öffnen eines Cursors:

- Speicherallokation für Cursor-Verarbeitung
- Parsen des SELECT-Statements
- Ausführung der SELECT-Anfrage
- Speicherung der Ergebnisdatensätze im CURSOR-Bereich
- Setzen des Cursor-Zeigers auf den ersten Datensatz

Syntax:

```
OPEN cursor_name;
```

Wenn die Anfrage keine Zeilen zurückliefert, wird eine Exception erzeugt.

Beispiel:

```
OPEN emp_cursor;
```

# Cursor-Daten lesen

Aktivitäten beim FETCH:

- lese die aktuellen Zeilenwerte in Ausgabevariablen
- gleiche Anzahl von Variablen wie Spalten in der Cursor-Definition
- jede Variable muss mit den Spalten von der Position her übereinstimmen

Syntax:

```
FETCH cursor_name INTO {variable1[, variable2, …] | record_name};
```

Beispiel:

```
FETCH emp_cursor INTO v_ename, v_sal;
```

- nach jedem FETCH rückt der Cursor-Zeiger automatisch einen Datensatz weiter
- Test, ob der Cursor Zeilen enthält mittels Cursor-Attributen

# Schließen des Cursors

Aktivitäten beim Schließen eines Cursors

- Schließen nach beendeter Verarbeitung der Zeilen
- Freigabe des allokierten Speichers
- anschließend kein Zugriff mehr auf Daten möglich, Re-Open des Cursors erforderlich

Syntax:

```
CLOSE cursor_name;
```

Bei Auftreten eines Fehlers bzgl. der durchgeführten Aktivitäten während des Schließens: Exception-Behandlung oder Sprung zur Aufrufumgebung.

# Explizite Cursor-Attribute

Attribute, die Aufschluss über den Status des Cursor geben (Metadaten)

| Attribut | Typ | Beschreibung |
|---|---|---|
| %ISOPEN | BOOLEAN | ist gleich TRUE, wenn der Cursor offen ist, ansonsten FALSE |
| %FOUND | BOOLEAN | ist gleich TRUE, wenn der letzte FETCH eine Zeile geliefert hat |
| %NOTFOUND | BOOLEAN | ist gleich TRUE, wenn der letzte FETCH überhaupt keine Zeile geliefert hat, Komplement zu %FOUND |
| %ROWCOUNT | NUMBER | Gesamtanzahl Zeilen, die bisher zurückgegeben wurden |

# Benutzerdefinierte Exceptions

Beispiel:

```
DECLARE
  e_invalid_product EXCEPTION;
BEGIN

  UPDATE product
  SET descrip = '&product_description'
  WHERE prodid = &product_number;

  IF SQL%NOTFOUND THEN
    RAISE e_invalid_product;
  END IF;
  COMMIT;

  EXCEPTION
    WHEN e_invalid_product THEN
      DBMS_OUTPUT.PUT_LINE('Invalid product number.');
END;
```

- Fehlerbehandlung in der WHEN-Klausel der Exception möglich (Protokollierung, Rollback)

# RAISE_APPLICATION_ERROR

Prozedur zur Ausgabe benutzerdefinierter Fehlermeldungen aus gespeicherten Subprogrammen

- Definition und Propagierung spezieller Fehler-Codes (SQLCODE) und –meldungen (SQLERRM), analog zu herkömmlichen Oracle-Fehlern
- Fehler-Codes im Bereich von -20001 bis -20999
- Genutzt an zwei Stellen: Ausführungsblock, Exception Handler

Syntax:

```
RAISE_APPLICATION_ERROR (error_number, message);
```

Beispiel:

```
IF LENGTH (p_jahr) <> 4  THEN
  RAISE_APPLICATION_ERROR(-20003,
                          'Jahreszahl fehlerhaft! Bitte 4-stellig eingeben');
END IF;
```