```
<div style="display: flex;
justify-content: center; flex-
direction: column; height:
70%"> <h1>PL/SQL</h1>
<h6>Presentation von
Michael Stenz</h6> </div>
```

Allgemein

 <li style="display: flex; align-items: center; list-style: none; position: relative;"> • Entwickelt von   Procedural Language extensions to SQL Seit Oracle 6.0 (1991) Erweiterung mit jeder Oracle Version

History

- **Pro*C** (< 1991): Oracle SQL Statements in C
- **PL/SQL 1.0** (1991): Sehr limitiert
- **PL/SQL 2.1**: Prozeduren, Funktionen, Packages
- **PL/SQL 2.2**: Calling Stored Functions in SQL => DBMS_SQL package & DDL statements
- **PL/SQL 2.3**: Binary PL/SQL programs
- **PL/SQL 2.4**: File I/O Support & PL/SQL Table/Record improvements (wie Arrays)
- **Oracle 19c**: Polymorphic Table Functions in the same package -> USING optional

Warum PL/SQL?

- Erweiterung durch 3GL-Elemente (Schleifen, Bedingungen, Prozeduren, Funktionen...)
- Auslagerung von Code/Business-logic in die Datenbank (Datenintegrität 👍, Keine Redundanz 👍)
- Vorbereitung und Speicherung Code (Precompiled im Datenbankcache ⚡)
- Zeit und Aktionsgesteuerte Ausführung (Trigger, Scheduler)
- Komplexe probleme -> einfache sub-programme
- ...

Architektur

- **PL/SQL Engine** ->
  Komponente von Oracle die PL/SQL Blöcke ausführt
- Engine führt proceduralen Code aus und sendet SQL Statements an den Database Server

Blockstruktur

```
<< label >> (optional)
DECLARE    -- Declarative part (optional)
-- Declarations of local types, variables, & subprograms

BEGIN      -- Executable part (required)
-- Statements (which can use items declared in declarative part)

[EXCEPTION -- Exception-handling part (optional)
-- Exception handlers for exceptions (errors) raised in executable part]
END;
```

 <h1>Basics & Datenstrukturen</h1>

Blockstruktur

Anonymer Block (nicht benannt)

```
BEGIN
-- Statements
END;
```

Blockstruktur

Lokale Prozedur (nur innerhalb des Blocks)

```
DECLARE
    PROCEDURE my_proc IS
    BEGIN
        -- Statements
    END;
BEGIN
    my_proc; -- Nur innerhalb des Blocks aufrufbar
END;
```

Blockstruktur

Stored Procedure (in DB gespeichert)

```
CREATE OR REPLACE PROCEDURE my_proc IS
BEGIN
    -- Statements
END;
```

Variablen

```
identifier [CONSTANT] datentyp [NOT NULL] [:= | DEFAULT ausdruck]
```

```
DECLARE
  emp_count NUMBER(3) := 0;
  part_no NUMBER(4);
  in_stock BOOLEAN;
BEGIN
    select count(*) into emp_count from emp; --Variablen mittels statements zuweisen
END;
```

%Type

Datentyp einer Spalte oder Variable
Verhindert Probleme bei Änderungen

```
DECLARE
  v_name emp.e_vname%TYPE
  n_name.emp.e_nname%TYPE
  tmpstr v_name%TYPE
BEGIN
```

DEMO

%RowType

Datentyp einer Zeile einer Tabelle (Cursor)

```
DECLARE
    dept_row dept%ROWTYPE;
BEGIN
    SELECT * INTO dept_row FROM dept WHERE deptno = 10;
END;
```

Records

Kann mehrere Variablen
verschiedener Datentypen
speichern
Deklaration mit %ROWTYPE
oder explizit

Datentypen

- SQL Types + PL/SQL Types
  - z.B BOOLEAN
- Scalar (können subtypes haben)
- Composite
- Large Object (LOB)

Subtype

- Nur subset von von Basis Typ

```
SUBTYPE subtype_name IS base_type
    { precision [, scale ] | RANGE low_value .. high_value } [ NOT NULL ]
```

DEMO

 <h1>Kontrollstrukturen</h1>

IF-THEN-ELSIF

Klassische verzweigungen

```
IF <condition1> THEN
  <sequence_of_statements1>
ELSIF <condition2> THEN
  <sequence_of_statements2>
ELSE
  <sequence_of_statements3>
END IF;
```

CASE

Vergleichbar mit IF-ELSE - wird effizienter ausgeführt.
Hierbei wird der selector verwendet anstatt bool'schen Ausdrücken

```
CASE selector
  WHEN expression1 THEN
    sequence_of_statements1;
  WHEN expression2 THEN
    sequence_of_statements2;
  ...
  [ELSE
    sequence_of_statementsN;]
END CASE;
```

Searched CASE

Ohne Selector

```
CASE
  WHEN grade = 'A' THEN dbms_output.put_line('Excellent');
  WHEN grade = 'B' THEN dbms_output.put_line('Very Good');
  WHEN grade = 'C' THEN dbms_output.put_line('Good');
  WHEN grade = 'D' THEN dbms_output.put_line('Fair');
  WHEN grade = 'F' THEN dbms_output.put_line('Poor');
  ELSE dbms_output.put_line('Invalid grade');
END CASE;
```

Case als Ausdruck

```
grade := CASE
  WHEN score >= 90 THEN 'A'
  WHEN score >= 80 THEN 'B'
  WHEN score >= 70 THEN 'C'
  WHEN score >= 60 THEN 'D'
  ELSE 'F'
END;
```

Loop

Endlosschleife

Benötigt **EXIT;** statement zum beenden.

```
LOOP
    -- statements
    EXIT WHEN <condition>; -- Gleich wie If-Then EXIT;
END LOOP;
```

Benennung von Schleifen

Schleifen können bennennt werden um mit **EXIT** oder **CONTINUE** gezielt zu springen.

```
<<loop1>>
FOR i IN 1..10 LOOP
  <<loop2>>
  FOR j IN 1..10 LOOP
    EXIT loop1 WHEN <condition>;
  END loop;
END loop;
```

While

```
WHILE <condition> LOOP
    -- statements
END LOOP;
```

For-Loop

Die zähler variable ist ein Integer, und muss nicht deklariert werden.

```
[<<loop_name>>]
FOR <counter> IN [REVERSE] <lower_bound>..<higher_bound> LOOP
  <sequence_of_statements> -- Counter kann innerhalb des loops nicht zugewiesen werden.
END LOOP;
```

```
FOR i IN 1..10 LOOP
  dbms_output.put_line(i);
END LOOP;
```

 <h1>Cursor</h1>

Quellen

- https://www.oracle.com/database/technologies/appdev/plsql.html
- https://docs.oracle.com/
- PL/SQL Basicscriptum - Mag. Johannes Tumfart
- Einführung PL/SQL - Mag. Johannes Tumfart