

PL/SQL

Allgemeiner Block

```
variable host_variable TYPE;

DECLARE
    TYPE type_name IS RECORD (
        member_name TYPE
        ...
    );

    TYPE type_name IS TABLE OF other_type [INDEX BY another_type];

    var_name          TYPE;
    var_name_init TYPE := initial_value;
    var_constraint TYPE NOT NULL := initial_value;
    var_const          CONSTANT TYPE := initial_value;

    import_var_type          tablename.columnname%TYPE;
    import_var_rowtype       tablename%ROWTYPE;

    CURSOR cursor_name IS select_statement [FOR UPDATE];
    record_name TYPE(of cursor);

    exception_name EXCEPTION;
BEGIN
    var_name = :host_variable;

    OPEN cursor_name;
    LOOP
        FETCH cursor_name INTO record_name;
        EXIT WHEN cursor_name%NOTFOUND;
    END LOOP;
    CLOSE cursor_name;

    FOR record_name2 IN cursor_name LOOP
        query WHERE CURRENT OF cursor_name
    END LOOP;

    FOR var IN a..b LOOP
    END LOOP;

    WHILE condition LOOP
    END LOOP;

    IF condition THEN
    [ELSIF condition2 THEN]
    [ELSE]
    END IF;

    CASE selector
        WHEN value1 THEN ...;
        WHEN value2 THEN ...;
        ELSE ...
    END CASE;

    RAISE exception_name
EXCEPTION
    WHEN exception_type THEN ...;
    WHEN OTHERS THEN ...;
END;
/
```

Datentypen

CHAR, CHAR(n)
VARCHAR(n)
NUMBER, NUMBER(s), NUMBER(s,p)
DATE
BOOLEAN
BINARY_INTEGER, PLS_INTEGER
BINARY_FLOAT, BINARY_DOUBLE

Cursor Variablen

cursor_name%FOUND
cursor_name%NOTFOUND
cursor_name%ROWCOUNT
cursor_name%ISOPEN

Procedures

```
CREATE [OR REPLACE] PROCEDURE procedure_name (  
    parameter_name [IN | OUT | IN OUT] TYPE, ...)  
(AS | IS)  
    ... Declare Teil ...  
BEGIN  
    ...  
EXCEPTION  
    ...  
END;  
  
DROP PROCEDURE procedure_name;
```

Aufruf aus PL/SQL

```
DECLARE  
    parameter_value TYPE;  
BEGIN  
    procedure_name(parameter_value, ...);  
END;  
/
```

Aufruf aus SQL*PLUS

```
variable parameter_value TYPE;  
execute procedure_name(:parameter_value, ...);
```

Transaction Control

COMMIT, ROLLBACK, SAVEPOINT

Functions

```
CREATE [OR REPLACE] FUNCTION function_name (  
    parameter_name [IN | OUT | IN OUT] TYPE, ...)  
RETURN type  
(AS | IS)  
    ... Declare Teil ...  
BEGIN  
    ...  
    RETURN expression;  
EXCEPTION  
    ...  
END;  
  
DROP FUNCTION function_name;
```

Aufruf aus PL/SQL

```
DECLARE  
    return_value TYPE;  
BEGIN  
    return_value := function_name(parameter_value);  
END;  
/
```

Aufruf aus SQL*PLUS

```
variable return_value TYPE;  
execute :return_value := function_name(parameter_value);
```

Aufruf in SQL-Statement

```
SELECT ..., function_name(table.column) AS text FROM ... WHERE ...;  
SELECT ... FROM ... WHERE function_name(table.column) condition;
```

Nur IN Parameter, keine Transaction Control CMDs,
nur SQL Datentypen bei Return/Parametern, keine Sideeffects auf DB

Packages

```
CREATE [OR REPLACE] PACKAGE package_name AS
    ... variablen/cursor, typ, exception Deklarationen ...
    ... Prozedurkoepfe ...
    ... Funktionskoepfe ..
END;
/
```

```
DROP PACKAGE package_name;
```

```
CREATE [OR REPLACE] PACKAGE BODY package_name AS
    ... private deklarationen ...
    ... Prozedurimplementierungen ...
    ... Funktionsimplementierungen ...
END;
/
```

```
DROP PACKAGE BODY package_name;
```

Verwendung der Definition

```
package_name.function/procedure_name(parameters);
```

Trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name
    [BEFORE | AFTER]
    [INSERT | DELETE | UPDATE]
    ON table_name
    [REFERENCING OLD AS name | NEW AS name]
    [FOR EACH ROW]
    [WHEN condition]
{BODY}
```

1. BODY:

```
DECLARE
    ...
BEGIN
    ...
EXCEPTION
    ...
END;
```

2. BODY:

```
CALL procedure_name
```

Default fuer new und old sind: :NEW und :OLD

INSTEAD OF Trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name
    INSTEAD OF [INSERT | DELETE | UPDATE]
    [OF column_name]
    ON view_name
    [REFERENCING OLD AS name | NEW AS name]
    [FOR EACH ROW]
    [WHEN condition]
{BODY}
```

```
DROP TRIGGER trigger_name;
ALTER TRIGGER ...
```