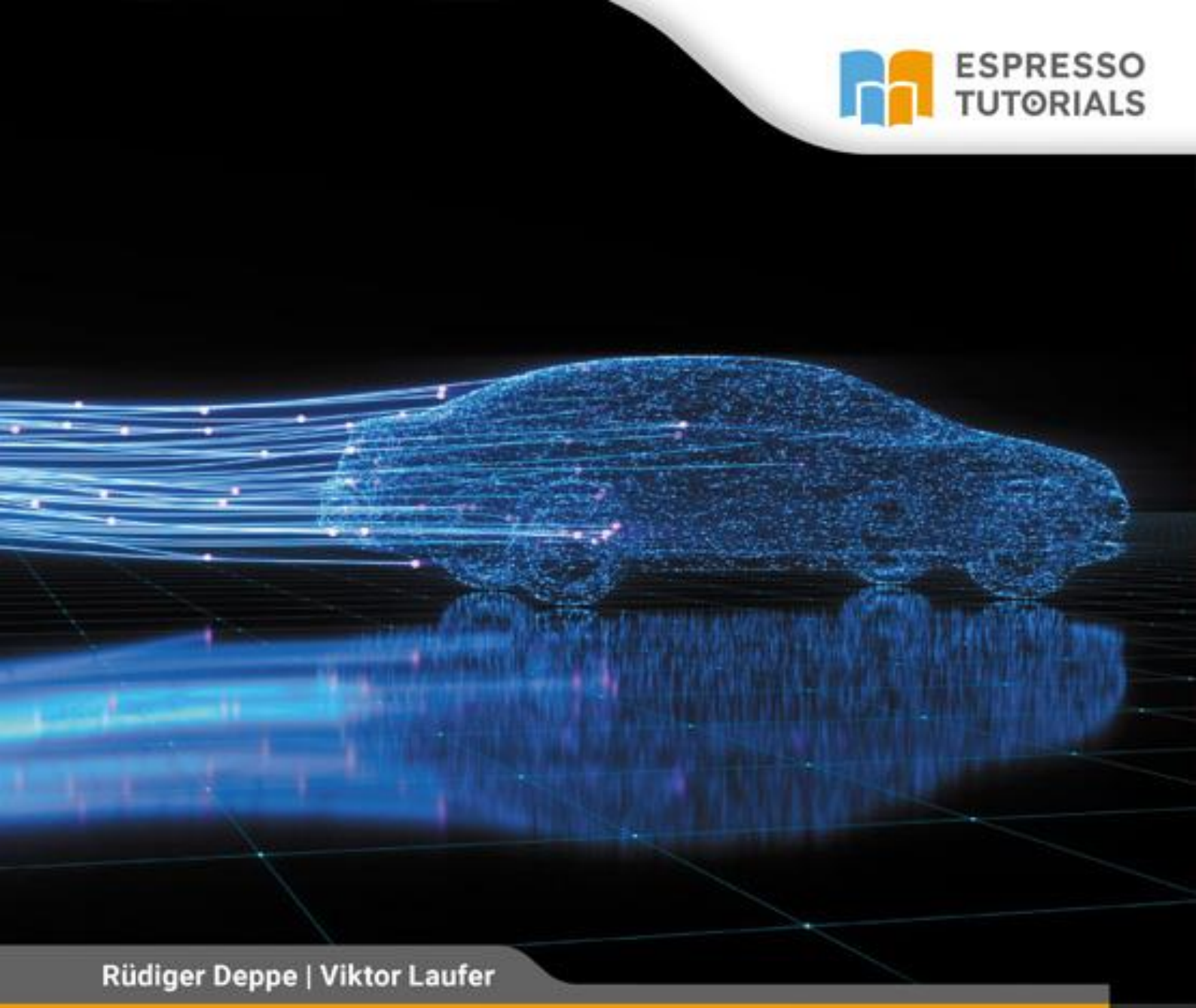
ESPRESSO TUTORIALS



Rüdiger Deppe | Viktor Laufer

# ABAP-Programmierung unter SAP S/4HANA®

2., erweiterte Auflage

- **Entwicklungsumgebung:** HANA-Objekte mit Eclipse entwickeln
- **Datenbankprogrammierung:** Sichten auf der SAP-HANA-Datenbank anlegen
- **Vergleich:** SQLScript versus die alten ABAP-Befehle
- **HANA-spezifische Techniken:** Textsuche und Entscheidungstabellen

# Sammlung der Code-Beispiel

## Abschnitt 1.3.5

```
CONSTANTS gc_lufthansa TYPE s_carr_id VALUE 'LH'.

SELECT carrid, connid, airpfrom, airpto
       FROM spfli
       INTO TABLE @DATA(gt_spfli)
       WHERE carrid = @gc_lufthansa.

LOOP AT gt_spfli
     ASSIGNING FIELD-SYMBOL(<gs_spfli>).
  WRITE: / <gs_spfli>-carrid,
           <gs_spfli>-connid,
           <gs_spfli>-airpfrom,
           <gs_spfli>-airpto.
ENDLOOP.
```

***Listing 1.1: Neues ABAP***

```
DATA gt_spfli TYPE TABLE OF spfli.
CONSTANTS gc_lufthansa TYPE s_carr_id VALUE 'LH'.
FIELD-SYMBOLS <gs_spfli> LIKE LINE OF gt_spfli.

SELECT carrid connid airpfrom airpto
       FROM spfli
       INTO CORRESPONDING FIELDS OF TABLE gt_spfli
       WHERE carrid = gc_lufthansa.

LOOP AT gt_spfli
     ASSIGNING <gs_spfli>.
  WRITE: / <gs_spfli>-carrid,
           <gs_spfli>-connid,
           <gs_spfli>-airpfrom,
           <gs_spfli>-airpto.
ENDLOOP.
```

***Listing 1.2: Altes ABAP***

## Abschnitt 1.3.6

```
SELECT spfli.carrid, spfli.connid, scarr.carrname,
       spfli.airpfrom, spfli.airpto
       FROM spfli JOIN scarr
       ON scarr.carrid = spfli.carrid;
```

***Listing 1.3: Selektion durch Native SQL***

## Abschnitt 2.2

```
REPORT zselektion_mit_adbc_neues_abap.

TYPES: BEGIN OF type_spfli,
         carrid       TYPE s_carr_id,
         connid       TYPE s_conn_id,
         airport_from TYPE s_fromairp,
         airport_to   TYPE s_toairp,
       END OF type_spfli.

DATA gt_spfli TYPE STANDARD TABLE OF type_spfli.
CONSTANTS gc_lufthansa TYPE s_carr_id VALUE 'LH'.

* Variablen füllen
DATA(gv_statement)
          = |SELECT carrid, connid, airpfrom, airpto |
         && |FROM spfli |
         && |WHERE mandt = { sy-mandt } |
         && |AND carrid = '{ gc_lufthansa }'|.

TRY.
* Select vorbereiten
    DATA(go_connection)
                = cl_sql_connection=>get_connection( ).
    DATA(go_statement)
                = go_connection->create_statement( ).

* Select durchführen
    DATA(go_result_set) = go_statement->execute_query(
                                        gv_statement ).
    go_result_set->set_param_table( REF #( gt_spfli ) ).
    go_result_set->next_package( ).
    go_result_set->close( ).
  CATCH cx_sql_exception INTO DATA(gx_exception).
    DATA(gv_error_message) = gx_exception->get_text( ).
    MESSAGE gv_error_message TYPE 'E'.
ENDTRY.

LOOP AT gt_spfli
     ASSIGNING FIELD-SYMBOL(<gs_spfli>).
  WRITE: / <gs_spfli>-carrid,
           <gs_spfli>-connid,
           <gs_spfli>-airport_from,
           <gs_spfli>-airport_to.
ENDLOOP.
```

***Listing 2.1: Selektion mit ADBC (neues ABAP)***

```
TYPES: BEGIN OF type_spfli,
         carrid       TYPE s_carr_id,
         connid       TYPE s_conn_id,
         airport_from TYPE s_fromairp,
         airport_to   TYPE s_toairp,
       END OF type_spfli.

DATA: gv_statement     TYPE string,
      go_connection    TYPE REF TO cl_sql_connection,
      go_statement     TYPE REF TO cl_sql_statement,
      go_result_set    TYPE REF TO cl_sql_result_set,
      gt_spfli         TYPE STANDARD TABLE OF type_spfli,
      gr_spfli         TYPE REF TO data,
      gx_exception     TYPE REF TO cx_sql_exception,
      gv_error_message TYPE string.
CONSTANTS gc_lufthansa TYPE s_carr_id VALUE 'LH'.
FIELD-SYMBOLS <gs_spfli> LIKE LINE OF gt_spfli.

* Variablen füllen
```

```
CONCATENATE 'SELECT carrid, connid, airpfrom, airpto'
            'FROM spfli'
            'WHERE mandt ='sy-mandt'
            'AND carrid = '''
            INTO gv_statement
            SEPARATED BY space.

CONCATENATE gv_statement
            gc_lufthansa
            ''''
            INTO gv_statement.

GET REFERENCE OF gt_spfli INTO gr_spfli.

TRY.
* Select vorbereiten
    go_connection = cl_sql_connection=>get_connection( ).
    go_statement  = go_connection->create_statement( ).

* Select durchführen
    go_result_set = go_statement->execute_query(
                                  gv_statement ).
    go_result_set->set_param_table( gr_spfli ).
    go_result_set->next_package( ).
    go_result_set->close( ).
  CATCH cx_sql_exception INTO gx_exception.
    gv_error_message = gx_exception->get_text( ).
    MESSAGE gv_error_message TYPE 'E'.
ENDTRY.

LOOP AT gt_spfli
     ASSIGNING <gs_spfli>.
  WRITE: / <gs_spfli>-carrid,
           <gs_spfli>-connid,
           <gs_spfli>-airport_from,
           <gs_spfli>-airport_to.
ENDLOOP.
```

***Listing 2.2: Selektion mit ADBC (altes ABAP)***

```
REPORT zloeschen_mit_adbc.

TRY.
* Verbindung vorbereiten
   DATA(go_connection)
         = cl_sql_connection=>get_connection( ).
   DATA(go_statement)
         = go_connection->create_statement( ).

* Daten löschen
   DATA(gv_cnt) = go_statement->execute_update( 'DELETE
                  FROM scarr' ).
 CATCH cx_sql_exception INTO gx_exception.
   DATA(gv_error_message) = gx_exception->get_text( ).
   MESSAGE gv_error_message TYPE 'E'.
ENDTRY.

COMMIT CONNECTION.
```

***Listing 2.3: Daten löschen und neu einfügen mit ADBC (neues ABAP)***

```
SELECT spfli~carrid, spfli~connid, scarr~carrname,
       spfli~airpfrom, spfli~airpto
       FROM spfli JOIN scarr
       ON scarr~carrid = spfli~carrid.
```

***Listing 2.4: JOIN-Select mit ABAP SQL***

```
SELECT spfli.carrid, spfli.connid, scarr.carrname,
       spfli.airpfrom, spfli.airpto
       FROM spfli JOIN scarr
       ON scarr.carrid = spfli.carrid;
```

***Listing 2.5: JOIN-Select mit nativem SQL***

## Abschnitt 2.3

```
SELECT carrid connid
       INTO ls_spfli
       FROM spfli
       CONNECTION ('SECOND')
       WHERE carrid = 'LH'.
```

***Listing 2.6: Sekundäre Datenbankverbindung mit ABAP SQL***

```
    go_connection = cl_sql_connection=>get_connection(
                      'SECOND' ).
    go_statement  = go_connection->create_statement( ).
```

***Listing 2.7: Sekundäre Datenbankverbindung mit ADBC***

## Abschnitt 2.4.1

```
CREATE COLUMN TABLE "ZAIRPLANE"
("MANDT" NVARCHAR(3) DEFAULT '000' NOT NULL,
"TYPE" NVARCHAR(6) DEFAULT '' NOT NULL,
"MODEL" NVARCHAR(5) DEFAULT '' NOT NULL,
"SEATS" SMALLINT,
"MOTOR" SMALLINT,
PRIMARY KEY ("MANDT", "TYPE", "MODEL"));
```

***Listing 2.8: Dynamisches Anlegen einer Datenbank-Tabelle***

```
CREATE VIEW zflight
AS SELECT sflight.mandt, sflight.carrid, sflight.connid,
          sflight.fldate, sbook.customid
          FROM sapdea.sflight INNER JOIN sapdea.sbook
          ON sflight.carrid = sbook.carrid
          AND sflight.connid = sbook.connid
          AND sflight.fldate = sbook.fldate;
```

***Listing 2.9: Dynamisches Anlegen eines Datenbank-Views***

## Abschnitt 2.4.2

```
SELECT CONCAT(carrid, connid) AS "Verbindung",
       passname AS "Passagier",
       to_date(fldate) AS "Flugdatum",
       to_date(order_date) AS "Buchungsdatum",
       CONVERT_CURRENCY(amount =>loccuram,
                    "SOURCE_UNIT_COLUMN"=>loccurkey,
                    "SCHEMA"            =>'SAPDEA',
                    "TARGET_UNIT_COLUMN"=>'EUR',
                    "REFERENCE_DATE" =>current_utcdate,
                    "CLIENT"=>'100') AS "Preis (EUR)"
       FROM sapdea.sbook
       WHERE DAYS_BETWEEN(order_date, fldate) < 100
         AND contains(passname, 'Max Mustermann',
                      FUZZY(0.8))
       ORDER BY fldate DESC
       LIMIT 10;
```

***Listing 2.10: SELECT-Anweisung in Native SQL in SAP HANA***

## Abschnitt 2.5

```
et_flights =
SELECT a.carrid, a.connid, a.fldate, b.customid
     FROM :it_sflights AS a INNER JOIN sapdea.sbook AS b
     ON a.mandt = b.mandt
     AND a.carrid = b.carrid
     AND a.connid = b.connid
     AND a.fldate = b.fldate
     WHERE a.mandt = :iv_mandt;
```

***Listing 2.11: Einfache Selektion mit SQLScript***

```
et_flights =
SELECT a.carrid, a.connid, a.fldate, b.customid
       FROM ( SELECT TOP 10 c.carrid, c.connid,
                       SUM(d.seatsocc)
              FROM sapdea.spfli AS c
              INNER JOIN sapdea.sflight AS d
              ON c.mandt = d.mandt
              AND c.carrid = d.carrid
              AND c.connid = d.connid
              WHERE c.mandt = :iv_mandt
              AND c.carrid = 'LH'
              GROUP BY a.carrid, a.connid
              ORDER BY SUM(d.seatsocc) DESC ) AS a
      INNER JOIN sapdea.sbook AS b
      ON a.mandt = b.mandt
      AND a.carrid = b.carrid
      AND a.connid = b.connid
      AND a.fldate = b.fldate
      WHERE a.mandt = :iv_mandt;
```

***Listing 2.12:  Umfangreiche Selektion mit SQL-Script***

```
ET_FLIGHTS =
SELECT carrid, connid, CASE WHEN fltime < '120000'
                            THEN 'Early'
                            ELSE 'Late'
                       END AS time
       FROM sapdea.spfli
       WHERE a.mandt = :iv_mandt;
```

***Listing 2.13: Fallunterscheidung in Selektionskriterien***

## Abschnitt 2.5.1

```
CREATE TYPE <type_name> AS TABLE (<column_definition>[{,< column_definition>}...])
```

***Listing 2.14: Anweisung zum Anlegen eines Tabellentyps***

```
CREATE TYPE tt_flights AS TABLE (
   carrid NVARCHAR(3),
   connid NVARCHAR(4));
```

***Listing 2.15: Anlage des Tabellentyps TT_FLIGHTS***

```
CREATE [OR REPLACE] PROCEDURE <proc_name> [(<parameter_clause>)] [LANGUAGE <lang>]
[SQL SECURITY <mode>] [DEFAULT SCHEMA <default_schema_name>]
 [READS SQL DATA [WITH RESULT VIEW <view_name>]]
 [WITH ENCRYPTION] AS
 BEGIN [SEQUENTIAL EXECUTION]
   <procedure_body>
 END;
```

***Listing 2.16: Anweisung zum Anlegen einer Datenbankprozedur***

```
CREATE PROCEDURE ProcWithResultView(
       IN iv_luggweight DECIMAL(8,4),
       OUT es_sbook SAPDEA.SBOOK)
 LANGUAGE SQLSCRIPT
 READS SQL DATA WITH RESULT VIEW ProcView AS
 BEGIN
   es_sbook = SELECT *
                    FROM SAPDEA.SBOOK
                    WHERE mandt = '100'
                      AND luggweight = :iv_luggweight;
 END;
```

***Listing 2.17: Beispiel-Anlage einer Datenbankprozedur***

```
CREATE PROCEDURE flights(
   IN iv_mandt NVARCHAR(3),
   IN iv_carrid NVARCHAR(3),
   OUT et_flights RDEPPE.TT_FLIGHTS ) AS
BEGIN
  lt_sbook = SELECT carrid, connid, customid
                    FROM sapdea.sbook
                    WHERE mandt  = :iv_mandt
                      AND carrid = :iv_carrid;

  et_flights = SELECT carrid, connid
                       FROM :lt_sbook
                       WHERE carrid = 'LH';
END;
```

***Listing 2.18: Selektion mit Eingabe- und Ausgabeparametern***

```
CREATE PROCEDURE flights_1(
   IN iv_mandt NVARCHAR(3),
   IN iv_carrid NVARCHAR(3),
   OUT et_flights RDEPPE.TT_FLIGHTS ) AS
BEGIN
  DECLARE lv_lines_sbook INTEGER;
  DECLARE lv_lines_flights INTEGER;
  DECLARE lv_lines_total INTEGER;
  DECLARE lv_count INTEGER;

  lt_sbook = SELECT carrid, connid, customid
                    FROM sapdea.sbook
                    WHERE mandt  = :iv_mandt
                      AND carrid = :iv_carrid;
  SELECT count(*)
         INTO lv_lines_sbook
         FROM :lt_sbook;

  et_flights = SELECT carrid, connid
                       FROM :lt_sbook
                       WHERE carrid = 'LH';
  SELECT count(*)
         INTO lv_lines_flights
         FROM :et_flights;

  lv_lines_total := :lv_lines_sbook + :lv_lines_flights;

  IF :lv_lines_total > 10
  THEN lv_lines_total := 4;
  ELSEIF :lv_LInes_total > 5
  THEN lv_lines_total := 3;
  ELSE lv_lines_total := 2;
  END IF;
END;
```

***Listing 2.19: Arbeiten mit Variablen in Datenbankprozeduren***

```
CREATE PROCEDURE flight_price(
       IN iv_mandt NVARCHAR(3),
       IN iv_carrid NVARCHAR(3),
       IN iv_connid NVARCHAR(4),
       OUT ev_price DECIMAL(15,2) )
 LANGUAGE SQLSCRIPT
 READS SQL DATA AS
 BEGIN
   SELECT price
          INTO ev_price
          FROM sapdea.sflight
          WHERE mandt  = :iv_mandt
            AND carrid = :iv_carrid
```

```
            AND connid = :iv_connid
            AND fldate = '19950228';

  lt_sbook = SELECT carrid, connid, customid
                    FROM sapdea.sbook
                    WHERE mandt  = :iv_mandt
                      AND carrid = :iv_carrid;
 END;
```

***Listing 2.20: Unterschied der Selektion nach skalaren Variablen und Tabellen***

```
PROCEDURE "_SYS_BIC"."Test_XS_Project::Determine_flights" ( )
LANGUAGE SQLSCRIPT
SQL SECURITY INVOKER
--DEFAULT SCHEMA <default_schema_name>
READS SQL DATA AS
BEGIN
/*****************************
Write your procedure logic
 *****************************/
END;
```

***Listing 2.21: Editor der Datenbankprozedur***

```
PROCEDURE "_SYS_BIC"."Test_XS_Project::Determine_flights"
( IN iv_mandt NVARCHAR(3),
  IN iv_carrid NVARCHAR(3),
  OUT et_flights TABLE ( carrid NVARCHAR(3),
                         connid NVARCHAR(4)
                       )
)
  LANGUAGE SQLSCRIPT
  SQL SECURITY INVOKER
  DEFAULT SCHEMA "SAPDEA"
  READS SQL DATA AS
BEGIN
  DECLARE lv_lines_sbook INTEGER;
  DECLARE lv_lines_flights INTEGER;
  DECLARE lv_lines_total INTEGER;

  lt_sbook = SELECT carrid, connid, customid
                    FROM sapdea.sbook
                    WHERE mandt  = :iv_mandt
                      AND carrid = :iv_carrid;
  SELECT count(*)
         INTO lv_lines_sbook
         FROM :lt_sbook;

  et_flights = SELECT carrid, connid
                      FROM :lt_sbook
                      WHERE carrid = 'LH';
  SELECT count(*)
         INTO lv_lines_flights
         FROM :et_flights;

  lv_lines_total := :lv_lines_sbook + :lv_lines_flights;

  IF :lv_lines_total > 10
  THEN lv_lines_total := 4;
  ELSEIF :lv_LInes_total > 5
  THEN lv_lines_total := 3;
  ELSE lv_lines_total := 2;
  END IF;
END;
```

***Listing 2.22: Beispiel für Datenbankprozedur***

## Abschnitt 2.5.2

```
CREATE FUNCTION determine_customer(
  Iv_mandt  NVARCHAR(3),
  iv_carrid NVARCHAR(3))
  RETURNS TABLE(
  Carrid   NVARCHAR(3),
  cityfrom NVARCHAR(20),
  cityto   NVARCHAR(20),
  customid NVARCHAR(8))
  LANGUAGE SQLSCRIPT SQL SECURITY INVOKER AS
BEGIN
  RETURN SELECT a.carrid, a.cityfrom, a.cityto,
                b.customid
               FROM sapdea.spfli AS a
                    INNER JOIN sapdea.sbook AS b
               ON a.mandt = b.mandt
               AND a.carrid = b.carrid
               AND a.connid = b.connid
               WHERE a.mandt  = :iv_mandt
                 AND a.carrid = :iv_carrid;
END;

SELECT c.carrid, c.cityfrom, c.cityto, c.customid, d.price
       FROM determine_customer('100', 'LH') AS c
       INNER JOIN sflight AS d
       ON  c.mandt = d.mandt
       AND c.carrid = d.carrid
       WHERE d.mandt = '100';
```

***Listing 2.23: Selektion mit integrierter benutzerdefinierter Funktion***

## Abschnitt 2.6.3

```
/********* Begin Procedure Script ************/
BEGIN
       var_out = SELECT price, currency
                 FROM "Test::AN_AIRLINE";
END /********* End Procedure Script ************/
```

***Listing 2.24: Calculation View mit SQLScript***

## Abschnitt 2.7.1

```
TYPES: BEGIN OF type_data,
         carrid    TYPE s_carr_id,
         connid    TYPE s_conn_id,
         city_from TYPE s_from_cit,
         city_to   TYPE s_to_city,
       END OF type_data.

DATA gt_data TYPE STANDARD TABLE OF type_data.
CONSTANTS: gc_lufthansa TYPE s_carr_id VALUE 'LH',
           gc_view TYPE string VALUE 'Test::AT_FLIGHT'.

* Variablen füllen
DATA(gv_statement)
          = |SELECT carrid, connid, cityfrom, cityto |
```

```
         && |FROM "{ gc_view }" |
         && |WHERE mandt = { sy-mandt } |
         && |AND carrid = '{ gc_lufthansa }'|.

TRY.
* Select vorbereiten
    DATA(go_connection)
                = cl_sql_connection=>get_connection( ).
    DATA(go_statement)
        = go_connection->create_statement( ).

* Select durchführen
    DATA(go_result_set) = go_statement->execute_query(
                                        gv_statement ).
    go_result_set->set_param_table( REF #( gt_data ) ).
    go_result_set->next_package( ).
    go_result_set->close( ).
  CATCH cx_sql_exception INTO DATA(gx_exception).
    DATA(gv_error_message) = gx_exception->get_text( ).
    MESSAGE gv_error_message TYPE 'E'.
ENDTRY.

LOOP AT gt_data
     ASSIGNING FIELD-SYMBOL(<gs_data>).
  WRITE: / <gs_data>-carrid,
           <gs_data>-connid,
           <gs_data>-city_from,
           <gs_data>-city_to.
ENDLOOP.
```

***Listing 2.25: ADBC für Attribut View AT_FLIGHT***

## Abschnitt 2.7.2

```
DATA gv_price TYPE s_price.
CONSTANTS: lc_mandt  TYPE s_mandt   VALUE '100',
           lc_carrid TYPE s_carr_id VALUE 'LH',
           lc_connid TYPE s_conn_id VALUE '0400'.

TRY.
    DATA(go_sql_statement) = cl_sql_connection=>get_connection( )->create_statement(
).
    go_sql_statement->set_param(
          data_ref = REF #( lc_mandt )
           inout    = cl_sql_statement=>c_param_in
           ).
    go_sql_statement->set_param(
          data_ref = REF #( lc_carrid )
          inout    = cl_sql_statement=>c_param_in
          ).
    go_sql_statement->set_param(
          data_ref = REF #( lc_connid )
          inout    = cl_sql_statement=>c_param_in
          ).
    go_sql_statement->set_param(
          data_ref = REF #( gv_price )
          inout    = cl_sql_statement=>c_param_out
          ).
    go_sql_statement->execute_procedure(
                               'rdeppe.flight_price' ).
  CATCH cx_root INTO DATA(lx_error).
    WRITE |{ lx_error->get_text( ) }|.
ENDTRY.

WRITE gv_price.
```

***Listing 2.26: Aufruf einer Datenbankprozedur ohne Strukturen oder Tabellen als Parameter***

```
TYPES: BEGIN OF type_flight,
         carrid TYPE s_carr_id,
         connid TYPE s_conn_id,
       END OF type_flight.
DATA: lv_statement TYPE string,
      lt_flight    TYPE TABLE OF type_flight.
CONSTANTS lc_carrid TYPE s_carr_id VALUE 'LH'.

TRY.
    lv_statement = |DROP TABLE #ET_FLIGHTS|.
    cl_sql_connection=>get_connection(
    )->create_statement( )->execute_ddl( lv_statement ).
  CATCH cx_sql_exception.
ENDTRY.

TRY.
    lv_statement = |CREATE LOCAL TEMPORARY ROW|
         && | TABLE #ET_FLIGHTS LIKE RDEPPE.TT_FLIGHTS|.
    cl_sql_connection=>get_connection(
    )->create_statement( )->execute_ddl( lv_statement ).

    lv_statement = |CALL "RDEPPE.FLIGHTS"|
                && |( '{ sy-mandt }', '{ lc_carrid }',|
                && | #ET_FLIGHTS ) WITH OVERVIEW|.
    DATA(lo_result_set) =
         cl_sql_connection=>get_connection(
            )->create_statement(
            )->execute_query( lv_statement ).
    lo_result_set->close( ).

    lv_statement = |SELECT * FROM #ET_FLIGHTS|.
    lo_result_set = cl_sql_connection=>get_connection(
                     )->create_statement(
                    )->execute_query( lv_statement ).

    lo_result_set->set_param_table(
                   REF #( lt_flight ) ).
    lo_result_set->next_package( ).
    lo_result_set->close( ).
  CATCH cx_sql_exception INTO DATA(lo_error).
    WRITE: | { lo_error->get_text( ) } |.
ENDTRY.

LOOP AT lt_flight
     ASSIGNING FIELD-SYMBOL(<ls_flight>).
  WRITE: / <ls_flight>-carrid,
           <ls_flight>-connid.
ENDLOOP.
```

***Listing 2.27: ABAP-Aufruf einer Datenbankprozedur mit Tabelle als Ausgangsparameter***

```
CREATE PROCEDURE flights_2(
   IN iv_mandt NVARCHAR(3),
   IN it_flights RDEPPE.TT_FLIGHTS,
   OUT ev_price DECIMAL(8,4) ) AS
BEGIN
/* Coding */
END;
```

***Listing 2.28: Datenbankprozedur mit Tabelle als Eingangsparameter***

```
TYPES: BEGIN OF type_flight,
         carrid TYPE s_carr_id,
         connid TYPE s_conn_id,
```

```
       END OF type_flight.
DATA: gv_statement TYPE string,
      gt_flight    TYPE TABLE OF type_flight,
      gv_price     TYPE s_price.

APPEND VALUE #( carrid = 'LH'
                connid = '0400' )
       TO gt_flight.

APPEND VALUE #( carrid = 'AA'
                connid = '3200' )
       TO gt_flight.

TRY.
    gv_statement = |CREATE LOCAL TEMPORARY ROW|
         && | TABLE #IT_FLIGHTS LIKE RDEPPE.TT_FLIGHTS|.
    cl_sql_connection=>get_connection(
                       )->create_statement(
                       )->execute_ddl( gv_statement ).

    LOOP AT gt_flight
         ASSIGNING FIELD-SYMBOL(<gs_flight>).
      gv_statement = |INSERT INTO #IT_FLIGHTS VALUES|
                  && |('{ <gs_flight>-carrid }',|
                 && | '{ <gs_flight>-connid }')|.
      cl_sql_connection=>get_connection(
                      )->create_statement(
                       )->execute_update( gv_statement ).
    ENDLOOP.

    gv_statement = | CALL "RDEPPE.FLIGHTS_2"|
                && |( '{ sy-mandt }', #IT_FLIGHTS,|
                && | '{ gv_price }') WITH OVERVIEW|.
    DATA(go_result_set) = cl_sql_connection=>get_connection(
        )->create_statement( )->execute_query( gv_statement ).
    go_result_set->close( ).
  CATCH cx_sql_exception INTO DATA(lo_error).
    WRITE: | { lo_error->get_text( ) } |.
ENDTRY.

WRITE gv_price.
```

***Listing 2.29: ABAP-Aufruf einer Datenbankprozedur mit Tabelle als Eingangsparameter***

```
TRY.
    CALL DATABASE PROCEDURE
         zdpp_flights
           EXPORTING
*            Parameter
           IMPORTING
*            Parameter
         .
  CATCH cx_sy_db_procedure_sql_error
        cx_sy_db_procedure_call INTO DATA(go_error).
    WRITE: | { go_error->get_text( ) } |.
ENDTRY.
```

***Listing 2.30: Aufruf eines Database Procedure Proxies aus ABAP***

## Abschnitt 2.9.2

```
@AbapCatalog.sqlViewName: 'sql_view_name'
@AbapCatalog.compiler.compareFilter: true
@AbapCatalog.preserveKey: true
```

```
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Flüge'
define view Zddl_Flight as select from data_source_name
left outer join joined_data_source_name
    on data_source_name.element_name = joined_data_source_name.joined_element_name {

}
```

***Listing 2.31: Anlage DDL Source und SQL-View***

```
@AbapCatalog.sqlViewName: 'ZSQL_DDL_FLIGHTS' @AbapCatalog.compiler.compareFilter:
true
@AbapCatalog.preserveKey: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Flüge'
define view Zddl_Flight as select from spfli
inner join sflight
    on  spfli.carrid = sflight.carrid
    and spfli.connid = sflight.connid {
  spfli.carrid,
  spfli.connid,
  spfli.cityfrom,
  spfli.cityto,
  sflight.fldate,
  sflight.price,
  sflight.currency
}
```

***Listing 2.32: Füllung des CDS-Views***

```
@AbapCatalog.sqlViewName: 'ZSQL_DDL_FLPLAN'
@AbapCatalog.compiler.CompareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Flugplan'
define view Zddl_Flightplan as select from sflight
association [1..1] to spfli as Connection
    on  sflight.carrid = Connection.carrid
    and sflight.connid = Connection.connid {
  carrid,
  connid,
  fldate,
  price,
  currency,
  planetype,
  seatsmax,
  seatsocc,
  Connection // Make association public
}
```

***Listing 2.33: Anlage eines Flugplans als CDS-View mit Assoziationen***

```
@AbapCatalog.sqlViewName: 'ZSQL_DDL_BOOKING'
@AbapCatalog.compiler.CompareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Buchung'
define view Zddl_Booking as select from sbook
association [1..1] to Zddl_Flightplan as Flightplan
    on  $projection.carrid = Flightplan.carrid
    and $projection.connid = Flightplan.connid
    and $projection.fldate = Flightplan.fldate {
  carrid,
  connid,
  fldate,
  bookid,
  customid,
  custtype,
  luggweight,
```

```
  wunit,
  class,
  loccuram,
  loccurkey,
  Flightplan // Make association public
}
```

***Listing 2.34: Anlage von Buchungen als CDS-View mit Assoziationen und Bezug zum Flugplan-CDS-View***

```
@AbapCatalog.sqlViewName: 'ZSQL_DDL_SELBOOK'
@AbapCatalog.compiler.CompareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'ausgewählte Buchungsdaten'
define view Zddl_Sel_Booking as select from Zddl_Booking as Booking{
  Booking.carrid,
  Booking.connid,
  Booking.Flightplan.Connection.cityfrom,
  Booking.Flightplan.Connection.cityto,
  Booking.fldate,
  Booking.Flightplan.price
}
```

***Listing 2.35: Anlage eines CDS-Views der aus dem Buchungs-CDS-View selektierten Buchungen***

```
@AbapCatalog.compiler.CompareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'ausgewählte Buchungsdaten'
define view Zddl_Sel_Booking_With_Param
with parameters p_carrid : abap.char(3),
                p_connid : abap.char(4)
as select from Zddl_Booking as Booking{
  Booking.carrid,
  Booking.connid,
  Booking.Flightplan.Connection.cityfrom,
  Booking.Flightplan.Connection.cityto,
  Booking.fldate,
  Booking.Flightplan.price
}
where Booking.carrid = $parameters.p_carrid
  and Booking.connid = $parameters.p_connid
```

***Listing 2.36: CDS-View mit Eingangsparametern***

```
SELECT *
       FROM Zddl_Sel_Booking_With_Param(
            p_carrid = 'LH',
            p_connid = '0400' )
       INTO TABLE @DATA(lt_booking).
```

***Listing 2.37: Selektionsaufruf für einen CDS-View***

```
SELECT *
       FROM Zddl_Sel_Booking_With_Param(
            p_carrid = 'LH',
            p_connid = '0400' )
       INTO TABLE @DATA(lt_booking)
       ##DB_FEATURE_MODE[VIEWS_WITH_PARAMETERS].
```

***Listing 2.38: Selektionsaufruf für CDS-View mit Meldungsunterdrückung***

```
@AbapCatalog.sqlViewName: 'ZSQL_DDL_FLPLAN'
@AbapCatalog.compiler.CompareFilter: true@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Flugplan'
define view Zddl_Flightplan as select from sflight
association [1..1] to spfli as Connection
    on  sflight.carrid = Connection.carrid
    and sflight.connid = Connection.connid {
  carrid,
  connid,
  fldate,
  Cast(
    Concat(
      Concat(
        Concat(substring(fldate, 5, 2), '.'),
        Concat(substring(fldate, 7, 2), '.')
      )
    Substring(fldate, 1, 4)
  ) As char10 preserving type ) as ZCONVERTED_DATE,
  price,
  currency,
  planetype,
  seatsmax,
  seatsocc,
  Connection // Make association public
}
```

***Listing 2.39: CDS-View mit Umwandlung des Datum-Formats***

```
@AbapCatalog.sqlViewName: 'ZSQL_DDL_BOCL'
@AbapCatalog.compiler.CompareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Buchungsklasse'
define view Zddl_Booking_Class
with parameters p_Booking_Class : abap.char( 1 )
as select from sbook as booking
inner join sflight as flight
    on booking.carrid = flight.carrid
    and booking.connid = flight.connid
    and booking.fldate = flight.fldate
{
    key booking.carrid,
    key booking.connid,
    key booking.fldate,
    case $parameters.p_Booking_Class
        when 'Y' //Economy Class
            then flight.seatsocc
        when 'C' //Business Class
            then flight.seatsocc_b
        else //First Class
            flight.seatsocc_f
    end as count_Occupied_Seats,
    count( * ) as count_Bookings
}
where booking.cancelled <> 'X'
  and booking.class = $parameters.p_Booking_Class
group by
    booking.carrid,
    booking.connid,
    booking.fldate,
    booking.class,
    flight.seatsocc,
    flight.seatsocc_b,
    flight.seatsocc_f
having
    //Occupied seats < number of bookings
    ( booking.class = 'Y'
      and flight.seatsocc < count( * ) )
    or ( booking.class = 'C'
       and flight.seatsocc_b < count( * ) )
    or ( booking.class = 'F'
```

```
        and flight.seatsocc_f < count( * ) )
```

***Listing 2.40: ABAP-CDS-View mit in die SELECT-Anweisung eingebauten Funktionen***

```
@AbapCatalog.sqlViewName: 'ZSQL_DDL_BOCH'
@AbapCatalog.compiler.CompareFilter: true
@AccessControl.authorizationCheck: #CHECK
@EndUserText.label: 'Buchungsprüfung'
define view Zddl_Booking_Check as
select from
    Zddl_Booking_Class( p_Booking_Class: 'Y' )
{
    key carrid,
    key connid,
    key fldate,
    cast('ECONOMY' as abap.char( 8 )) as class,
    count_Bookings,
    count_Occupied_Seats,
    abs(count_Bookings - count_Occupied_Seats)
        as difference
}
union all
    select from Zddl_Booking_Class( p_Booking_Class: 'C' )
{
    carrid,
    connid,
    fldate,
    'BUSINESS' as class,
    count_Bookings,
    count_Occupied_Seats,
    abs(count_Bookings - count_Occupied_Seats)
        as difference
}
union all
    select from Zddl_Booking_Class( p_Booking_Class: 'F' )
{
    carrid,
    connid,
    fldate,
    'FIRST' as class,
    count_Bookings,
    count_Occupied_Seats,
    abs(count_Bookings - count_Occupied_Seats)
        as difference
}
```

***Listing 2.41: CDS-View mit UNION ALL und anderen Funktionen***

```
@AbapCatalog.sqlViewName: 'DEMO_CDS_T100'
@AccessControl.authorizationCheck: #NOT_REQUIRED
define view demo_cds_select_t100
  as select from
    demo_cds_select_t100_langu(
      p_langu: $session.system_language )
    {
      *
    }
```

***Listing 2.42: CDS-View mit der Session-Variable system_language***

```
sbook.order_date,
    CEIL( sbook.luggweight ) as zzluggweight
}
```

***Listing 2.43: Erweiterung eines vorhandenen ABAP-CDS-Views***

```
@AbapCatalog.sqlViewName: 'ZSQL_ANNOTAT'
@AbapCatalog.compiler.CompareFilter: true
@AccessControl.authorizationCheck: #CHECK
@AbapCatalog.Buffering.status: #ACTIVE
@AbapCatalog.Buffering.type: #FULL
@EndUserText.label: 'Annotationen'
define view Zddl_Annotations as select from spfli {
    @EndUserText.label: 'Airline'
    key carrid,

    @EndUserText.label: 'Connection'
    key connid,

    @EndUserText.label: 'Country from'
    @Semantics.address.country
    countryfr,

    @EndUserText.label: 'City from'
    @Semantics.address.city
    cityfrom,

    @EndUserText.label: 'Country to'
    @Semantics.address.country
    countryfr,

    @EndUserText.label: 'City to'
    @Semantics.address.city
    cityto,

    @EndUserText.label: 'Flight Duration'
    @Semantics.calendarItem.duration
    fltime,

    @EndUserText.label: 'Distance'
    @Semantics.quantity.unitOfMeasure: 'distid'
    distance,

    @EndUserText.label: 'Measure of distance'
    @Semantics.unitOfMeasure: 'true'
    distid
}
```

***Listing 2.44: CDS-View mit Annotationen***

```
DATA(go_alv_display) = cl_salv_gui_table_ida=>
    create_for_cds_view( CONV #( 'ZDDL_ANNOTATIONS' ) ).
go_alv_display->fullscreen( )->display( ).

Alternative verkürzte Schreibweise:
cl_salv_gui_table_ida=>create( iv_table_name = 'ZDDL_ANNOTATIONS' )->fullscreen( )-
>display( ).
```

***Listing 2.45: Anzeige eines ABAP CDS-Views im ALV with IDA***

## Abschnitt 2.9.3

```
namespace Test;

@Schema: 'RDEPPE'
context CustomerSelection {

  type StreetAddress {
    name   : String(80);
    number : Integer;
```

```
  };

  type CountryAddress {
    name : String(80);
    code : String(3);
  };

  @Catalog.tableType : #COLUMN
  entity Address {
    key id  : Integer;
    street  : StreetAddress;
    zipCode : Integer;
    city    : String(80);
    country : CountryAddress;
    type    : String(10); // home, office
  };

  @Catalog.tableType : #COLUMN
  define entity Partner {
    key id      : Integer;
    addressId   : Integer;
    role        : String(3);
    companyName : String(80);
    _Address    : association[1] to Address
    on addressId = _Address.id;
   };

  define view Customer as select from Partner {
    key id,
    'CUSTOMER' as role,
    companyName,
    _Address.street,
    _Address.countryCode
    } where role = '03';

};
```

***Listing 2.46: Code einer HANA-CDS-Datei***

## Abschnitt 2.9.4

```
CLASS zcl_flights_amdp DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC.

  PUBLIC SECTION.
    INTERFACES: if_amdp_marker_hdb.

    TYPES: BEGIN OF type_flights,
             carrid TYPE s_carr_id,
             connid TYPE s_conn_id,
           END OF type_flights.
   TYPES tt_flights TYPE STANDARD TABLE OF type_flights.

    METHODS: get_flights
      IMPORTING
        VALUE(iv_mandt)   TYPE mandt
        VALUE(iv_carrid)  TYPE s_carr_id
      EXPORTING
        VALUE(et_flights) TYPE tt_flights.
ENDCLASS.



CLASS zcl_flights_amdp IMPLEMENTATION.
  METHOD get_flights
         BY DATABASE PROCEDURE
```

```
          FOR HDB LANGUAGE SQLSCRIPT
          OPTIONS READ-ONLY USING sbook.
    lt_sbook = SELECT carrid, connid, customid
                      FROM sbook
                      WHERE mandt  = :iv_mandt
                        AND carrid = :iv_carrid;

    et_flights = SELECT carrid, connid
                        FROM :lt_sbook
                        WHERE carrid = 'LH';
  ENDMETHOD.
ENDCLASS.
```

***Listing 2.47: AMDP-Klasse***

```
TRY.
    DATA(go_flights) = NEW zcl_flights_amdp( ).
    go_flights->get_flights(
      EXPORTING
        iv_mandt  = sy-mandt
        iv_carrid = 'AA'
      IMPORTING
        et_flights = DATA(gt_flights)
      ).
  CATCH cx_amdp_error INTO DATA(gx_error).
    WRITE |{ gx_error->get_text( ) }|.
ENDTRY.
```

***Listing 2.48: Aufruf einer AMDP-Methode aus einem ABAP-Programm***

```
METHOD call_get_carriers
       BY DATABASE PROCEDURE FOR HDB
       LANGUAGE SQLSCRIPT
       USING cl_demo_amdp_changing=>get_carriers.
   CALL "CL_DEMO_AMDP_CHANGING=>GET_CARRIERS"(
        CARRIERS__IN__  => :CARRIERS,
        CARRIERS        => :CARRIERS );
ENDMETHOD.
```

***Listing 2.49: Aufruf einer AMDP mit CHANGING-Parametern aus einer anderen AMDP***

```
METHOD get_carriers BY DATABASE PROCEDURE FOR HDB
                       LANGUAGE SQLSCRIPT
                     USING scarr.
   carriers  = SELECT s.*
                      FROM scarr AS s
                      INNER JOIN :carriers AS c
                      ON s.mandt  = c.mandt AND
                         s.carrid = c.carrid;
ENDMETHOD.
```

***Listing 2.50: Verarbeitung von CHANGING-Parametern bei AMDP***

## Abschnitt 3.1.2

```
REPORT zmaintain_full_text_index.

" Konfiguration
PARAMETERS:
  table  LIKE dd02l-tabname DEFAULT 'SCUSTOM',
  column LIKE dd03l-fieldname DEFAULT 'NAME',
  fzyidx TYPE abap_bool AS CHECKBOX DEFAULT abap_false,
  ta     TYPE abap_bool AS CHECKBOX DEFAULT abap_false,
  taconfig TYPE string DEFAULT 'EXTRACTION_CORE',
  drop   TYPE abap_bool AS CHECKBOX DEFAULT abap_true,
  create TYPE abap_bool AS CHECKBOX DEFAULT abap_true.

" Indexname (<Tabelle>~<Spalte>)
DATA(lv_idx) = table && '~' && column.
" SQL-Anweisung zum Anlegen eines Full-Text-Index
DATA(lv_sql) = |CREATE FULLTEXT INDEX { lv_idx } |
&& |ON { table }({ column })|.

" Zusätzlicher Fuzzy-Search-Index
IF fzyidx = abap_true.
  lv_sql = lv_sql && ' FUZZY SEARCH INDEX ON'.
ENDIF.

" Textanalyse
IF ta = abap_true.
  lv_sql = lv_sql && ' TEXT ANALYSIS ON'.
  " Spezielle Konfiguration der Textanalyse
  IF ( taconfig IS NOT INITIAL ).
    lv_sql = lv_sql && | CONFIGURATION '{ taconfig }'|.
  ENDIF.
ENDIF.

IF drop = abap_true.
  TRY.
      " Index entfernen
      cl_sql_connection=>get_connection(
      )->create_statement( )->execute_ddl(
      |DROP FULLTEXT INDEX { lv_idx }|
      ).
      WRITE: / |Fulltext index { lv_idx } entfernt|.
    CATCH cx_sql_exception INTO DATA(lo_ex).
      " Fehlerbehandlung
      WRITE: / | Fehler: { lo_ex->get_text( ) }|.
  ENDTRY.
ENDIF.

IF create = abap_true.
  TRY.
      " Textindex über ADBC anlegen
      cl_sql_connection=>get_connection(
      )->create_statement(
      )->execute_ddl( lv_sql ).
      WRITE: / |Fulltext index { lv_idx } angelegt|.
    CATCH cx_sql_exception INTO DATA(lo_ex1).
      " Fehlerbehandlung
      WRITE: / | Fehler: { lo_ex1->get_text( ) }|.
  ENDTRY.
ENDIF.
```

***Listing 3.1: Report für das Anlegen und Löschen eines Full-Text-Index***

```
SELECT *
       FROM scustom
       WHERE CONTAINS(name, 'Frauke', FUZZY(0.8));
```

***Listing 3.2: Textsuche über CONTAINS***

```
SELECT *
  FROM scustom
   WHERE CONTAINS((name, city), 'Frauke' OR Frank*, EXACT);
```

***Listing 3.3: CONTAINS mit mehreren Feldern und Suchbegriffen***

```
SELECT *
  FROM scustom
   WHERE CONTAINS(*, 'Frauke' OR Frank*, EXACT);
```

***Listing 3.4: CONTAINS mit * als Spaltenangabe***

```
SELECT *
       FROM scustom
       WHERE CONTAINS(name, 'Frauke', FUZZY(0.8))
       ORDER BY SCORE( ) DESC;
```

***Listing 3.5: Sortierung der gefundenen Texte nach Übereinstimmungsgrad zur Suchanfrage***

```
SELECT SCORE( ) AS score, name
       FROM scustom
       WHERE CONTAINS(name, 'Frauke', FUZZY(0.8))
       ORDER BY score DESC;
```

***Listing 3.6: Ausgabe des Übereinstimmungswertes zur Suchanfrage***

```
SELECT *, HIGHLIGHTED(name)
       FROM scustom
       WHERE CONTAINS(name, 'Frauke', FUZZY(0.8))
       ORDER BY SCORE( ) DESC;
```

***Listing 3.7: Textsuche mit hervorgehobener Funktion – Ausgabe ganzer Wörter***

```
SELECT *, SNIPPETS(name)
       FROM scustom
       WHERE CONTAINS(name, 'Frauke', FUZZY(0.8))
       ORDER BY SCORE( ) DESC;
```

***Listing 3.8: Textsuche mit hervorgehobener Funktion – Ausgabe von Wortteilen***

```
SELECT *
  FROM scustom
  WHERE CONTAINS(name, 'M*', FUZZY(0.8,
  'textSearch=compare, stopwordTable=ZSTOPPWORD,
   stopwordListId=Name, similarCalculationMode=search' )
  );
```

***Listing 3.9: Textsuche mit Stoppwörtern***

```
SELECT *
 FROM scustom
 WHERE CONTAINS(name, 'Müller', FUZZY(0.8,
 'textSearch=compare, termMappingTable=ZSYNONYM,
 termMappingListId=Name, similarCalculationMode=search')
 );
```

***Listing 3.10: Textsuche mit Synonymen***

```
create column table custom_fuzzy (
   mandt NVARCHAR(3) DEFAULT '000' NOT NULL ,
   id NVARCHAR(8) DEFAULT '00000000' NOT NULL ,
   name NVARCHAR(25) DEFAULT '' NOT NULL ,
   city NVARCHAR(25) DEFAULT '' NOT NULL ,
   postcode NVARCHAR(10) FUZZY SEARCH MODE 'postcode',
   lastbooking DATE
);
```

***Listing 3.11: Anlage einer HANA-Datenbanktabelle mit speziellen Datentypen für die Textsuche in Datum und Postleitzahl***

```
INSERT INTO custom_fuzzy
SELECT c.mandt, c.id, c.name, c.city, c.postcode,
       TO_DATE( MIN ( b.order_date ) ) AS lastbooking
FROM sapdea.sbook AS b INNER JOIN sapdea.scustom AS c
ON b.mandt = c.mandt AND b.customid = c.id
GROUP BY c.mandt, c.id, c.name, c.city, c.postcode;
```

***Listing 3.12: Füllen der Datums- und Postleitzahltabelle mit Daten***

```
SELECT lastbooking, SCORE()
       FROM custom_fuzzy
       WHERE CONTAINS(lastbooking, '2020-11-13',
             FUZZY(0.9, 'maxDateDistance=3'))
       ORDER BY SCORE() DESC;
```

***Listing 3.13: Selektion des Datums***

```
SELECT postcode, SCORE()
       FROM custom_fuzzy
       WHERE CONTAINS( postcode, '69190', FUZZY(0.7))
       ORDER BY SCORE() desc;
```

***Listing 3.14: Selektion der Postleitzahl***

```
SELECT *
       FROM scustom
       WHERE CONTAINS(name, 'Frauke', FUZZY(0.8,
       'similarCalculationMode=search'));
```

***Listing 3.15: Textsuche mit FUZZY-Parametern***

```
SELECT *
      FROM scustom
      WHERE
      CONTAINS(street, 'Linden',
        FUZZY(0.8,
            'similarCalculationMode=flexible,
             lengthTolerance=0.85,
             errorDevaluate=0.9,
             fuzzySubstringMatch=beginning'
        )
      )
      ORDER BY score DESC;
```

***Listing 3.16: Textsuche mit Funktion fuzzySubstringMatch***

## Abschnitt 3.2.2

```
MODULE pbo OUTPUT.
  " ALV anlegen mit externem View als Datenbasis
  DATA(lo_alv_display) = cl_salv_gui_table_ida=>create(
       iv_table_name = 'ZEV_FLIGHT_PRICE'
       io_gui_container = NEW cl_gui_custom_container(
                                lv_container ) ).

  " Initiale Sortierung
  lo_alv_display->default_layout( )->set_sort_order(
                 VALUE #( ( field_name = 'DISCOUNT_NEW'
                            is_grouped = abap_false
                            descending = abap_true ) )
  ).
ENDMODULE.
```

***Listing 3.17: Darstellung der Ergebnisse aus einer Entscheidungstabelle in einer ALV-Liste***