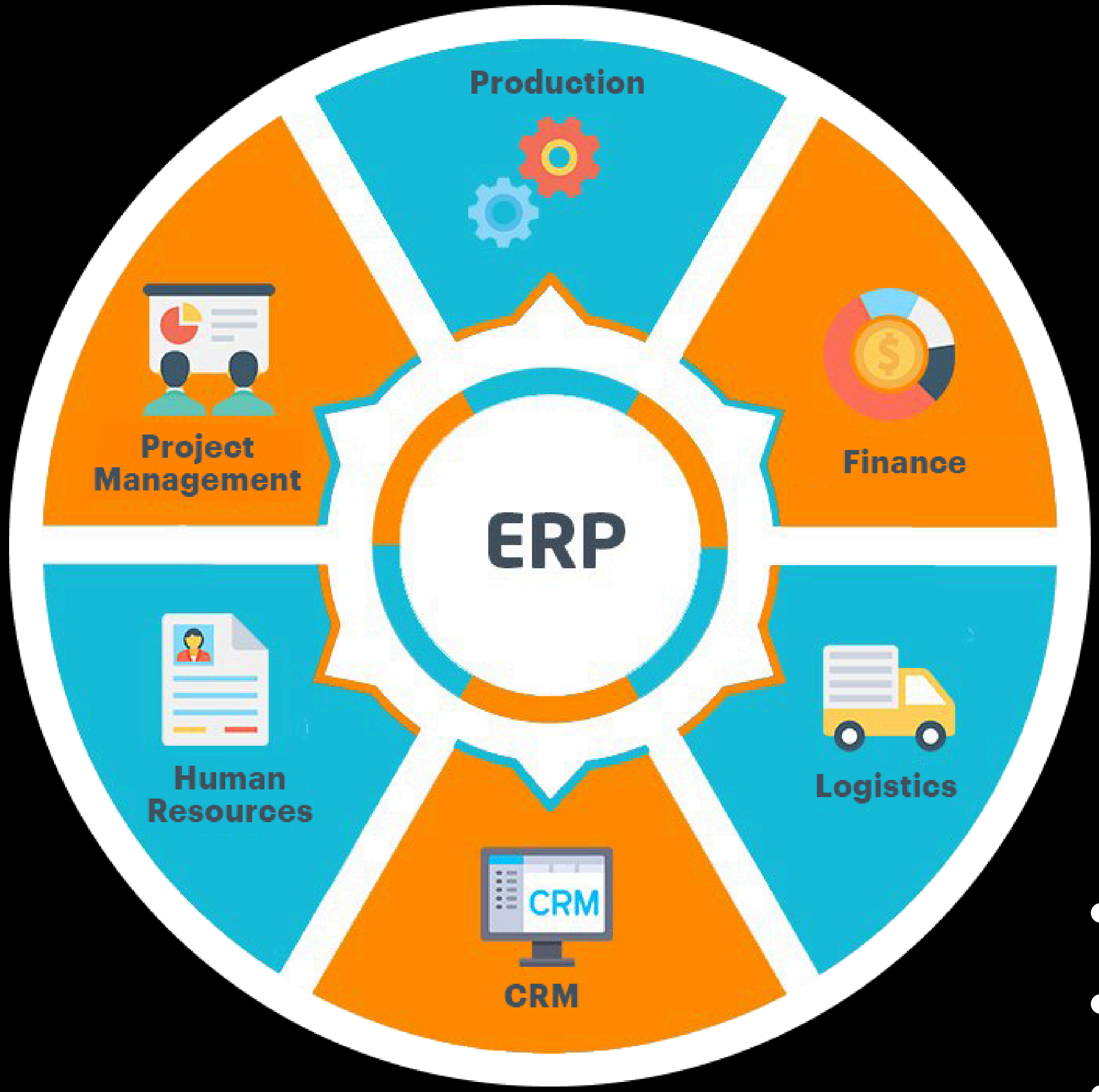
Internship Report

May 2025 - July 2025

Sharon Grace Prabhu
Arundhathi R

What is an ERP?

- Enterprise Resource Planning(ERP) is a system that is used to integrate various different operations of an enterprise into a single unified system.
- The main purpose of ERP systems is to streamline business processes thus enhancing data accuracy and improving the decision-making process.
- It enables organizations to manage complex operations efficiently and adapt to market changes.



KEY COMPONENTS OF ERP

ERP systems are typically composed of various interconnected modules each of which addresses a specific business function.

- Financial Management (FI)
- Human Resources(HR)
- Supply Chain Management(SCM)
- Procurement Planning(PP)
- Sales and Distribution(SD)
- Project Management(PM)
- Material Management(MM)

ADVANTAGES OF ERP

- Enhanced operational efficiency
- Improved decision making
- Cost savings
- Scalability and Flexibility
- Risk Management

INTRODUCTION TO SAP

- Systems, Applications and Products(SAP) is a German software corporation that develops enterprise software solutions to manage business operations.

- It was founded in 1972 by five former IBM employees: Dietmar Hopp, Hasso Plattner, Klaus Tschira, Claus Wellenreuther, Hans-Werner Hector

LAYERS IN SAP

1.PRESENTATION LAYER

- This is the topmost layer that directly interacts with the end-user.
- Its primary function is to provide the user interface through which users access and interact with the SAP system.

2. APPLICATION LAYER

- This is the layer that contains all the business logic, processes data and handles communication between the presentation and database layers.

3. DATABASE LAYER

- This is the lowest layer which is responsible for storing and managing all the data in the SAP system.
- It ensures data integrity, security and efficient data retrieval.

EVOLUTION OF SAP

1.SAP R/1:

- The presentation ,application and database layers are all resided on a single mainframe server.

2.SAP R/2:

- The presentation layer was given a separate server while the application and database layers still ran on a central mainframe.

3.SAP R/3:

- All the three layers run from three different servers.

4.SAP Business Suite:

The SAP Business Suite was introduced as a collection of integrated business applications in addition to core ERP services.

5.SAP ECC:

The SAP ECC is the refined version of the R/3 architecture which acts as the central ERP component within the SAP Business Suite.

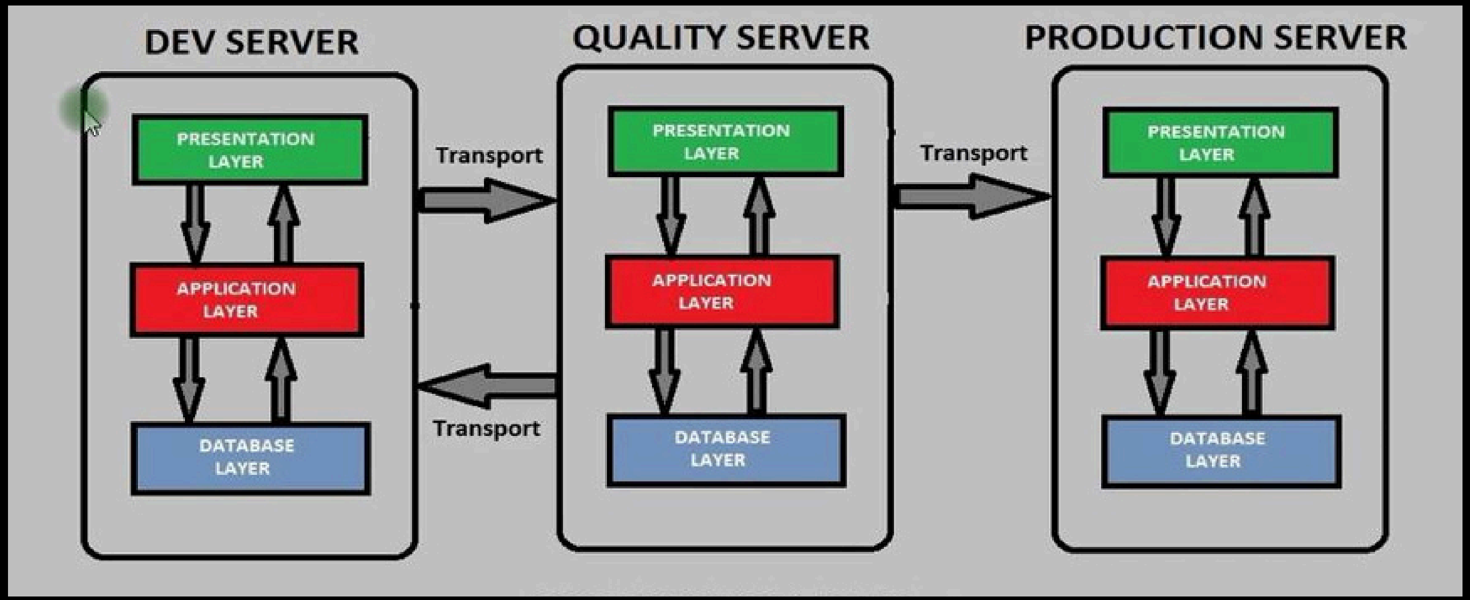
6.SAP S/4HANA:

This represents the next generation of SAP systems which exclusively uses the SAP HANA in-memory database.

INTRODUCTION TO ABAP

- The **Advanced Business Application Programming** or ABAP is SAP's proprietary programming language.
- It is primarily used for developing and customizing applications within the SAP environment.
- It can be used to customize SAP programs, create specialized reports and design custom forms.
- In essence, it is used for customization and adding unique features to SAP according to a company's need.

LANDSCAPE OF SAP



1.DEVELOPMENT SYSTEM :
This is where all the configuration and new developments (ABAP programs) are initially created.

2.QUALITY ASSURANCE SYSTEM :
This is where the testing is performed to ensure that developments and configurations work correctly and meet business requirements.

3.PRODUCTION SYSTEM :
This is the live system where actual business transactions are executed and real-time data is processed.

PACKAGES

- A package is a logical container in the SAP system used to group related development objects.
- It is used to organize development work in a structured and modular fashion thus facilitating transport across the different servers.
- It keeps development clean and manageable
- Promotes reusable components

TRANSPORT REQUEST

- Transport Request is a mechanism that is used to move development objects and their changes within SAP landscape.
- It records all the modifications made by developers to objects within a package.
- When objects are to be moved from one server to the other the transport requests are released from the present server and imported to the destination server.

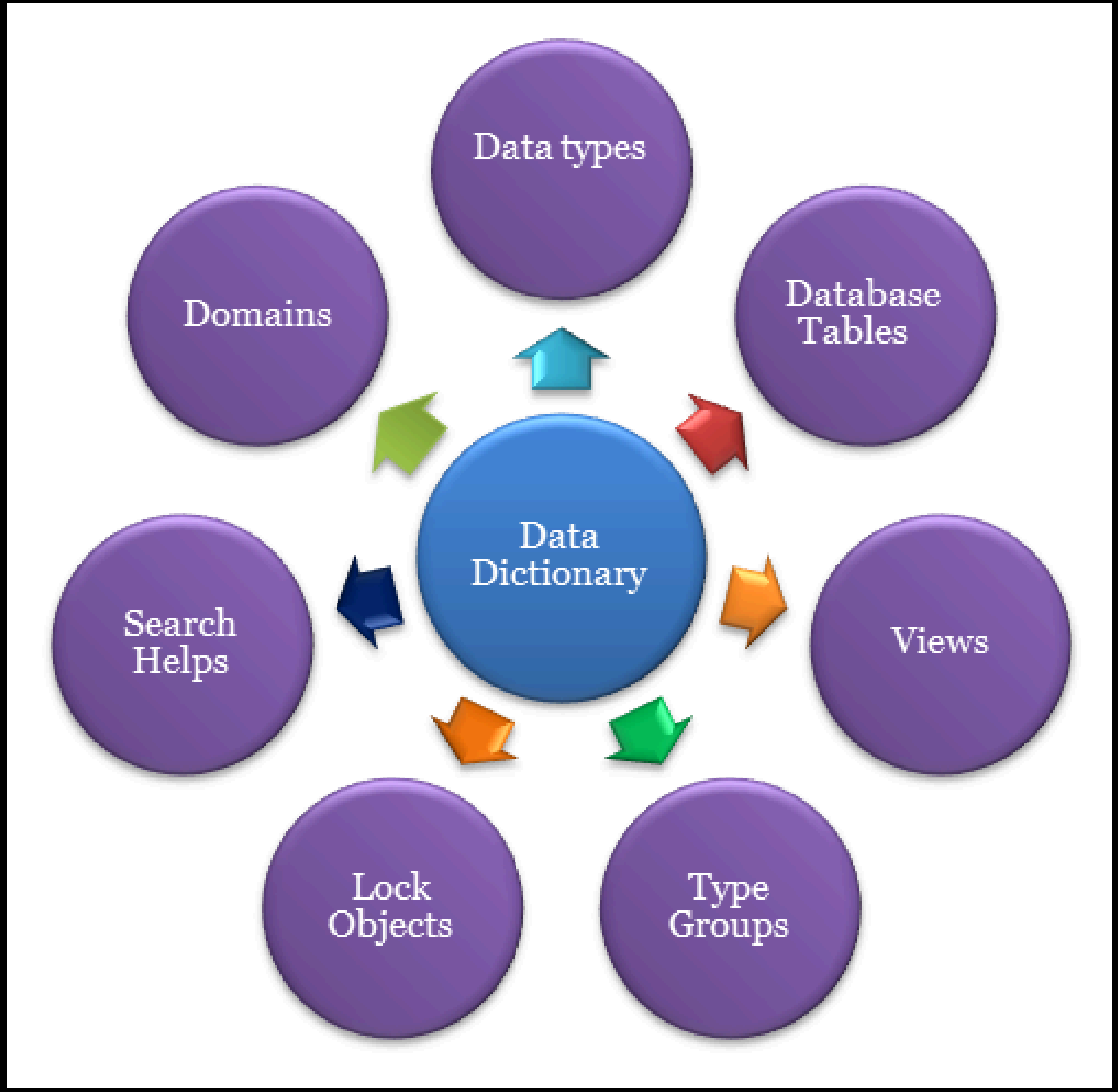
WRICEF

WRICEF is an acronym used to describe different types of custom developments that are needed to adapt the system to a company's specific business needs

- **W**orkflows : Automated processes for managing tasks and approvals within the system.
- **R**eports : Custom reports and queries for retrieving and displaying data.
- **I**nterfaces : Integration points between SAP and other systems, enabling data exchange.
- **C**onversions : Data migration and conversion tools.
- **E**nhancements : Modifications to existing SAP functionality to meet specific requirements.
- **F**orms : Custom forms and layouts for data entry and output.

DATA DICTIONARIES

- In SAP, a data dictionary (DDIC) is **a central repository for managing data definitions, also known as metadata.**
- It defines how data is structured, stored, and accessed within the SAP system, ensuring consistency and integrity.
- The DDIC acts as a structured source of information for creating and maintaining database-related objects like tables, domains, and data elements.



Components of DDIC

- **Domain** – Defines the technical attributes of a field such as data type, length, and value range.
- **Data Element** – Provides semantic meaning to a field and references a domain. → What value does it hold?
- **Table** – A physical database table used to store structured data with a primary key.
- **View** – A view is a "virtual table" containing fields from one or more tables.
- **Structure** – A reusable collection of fields grouped without a primary key.
- **Table Type** – Defines the structure and access method (standard, sorted, hashed) for internal tables.

Components of DDIC

Runtime Objects (Defined Using DDIC Types)

- **Internal Table** – A temporary memory table used in ABAP to hold **multiple records** during execution. (Table types)
- **Work Area** – A temporary buffer that holds a **single row** of data for processing in ABAP.(structures)

NOTE: Structures and Table types act as schemas/ definitions, whereas work areas and internal tables are used to assign the actual values.

Components of DDIC

- **Search help:** Provides F4 help — also known as input help or value help — when a user presses F4 on an input field. It suggests a list of valid values for that field, making data entry faster and error-free.

TYPES OF SEARCH HELP:

- **Elementary**: Elementary search helps implement a single search path for determining the possible entries

- **Collective**: Collective search helps contain several elementary search helps.

Search help

LPOS vs SPOS in Search Help

- LPOS (List Position):
  Defines the position of the field in the output list displayed to the user.

- SPOS (Search Position):
  Defines the position of the field in the selection screen
  (input field shown before search).

If SPOS = 0 → Field is not shown in the selection screen.
If LPOS = 0 → Field is not shown in the result list.

Hit List

- The Hit List is the output list of matching entries returned after a user performs a search using the search help.

- It displays all values that match the user’s input criteria (from the selection screen).

- Users can select a value from the hit list, and that selected value is returned to the input field in the calling screen.

DATA TYPES

Data types are templates for creating data objects.

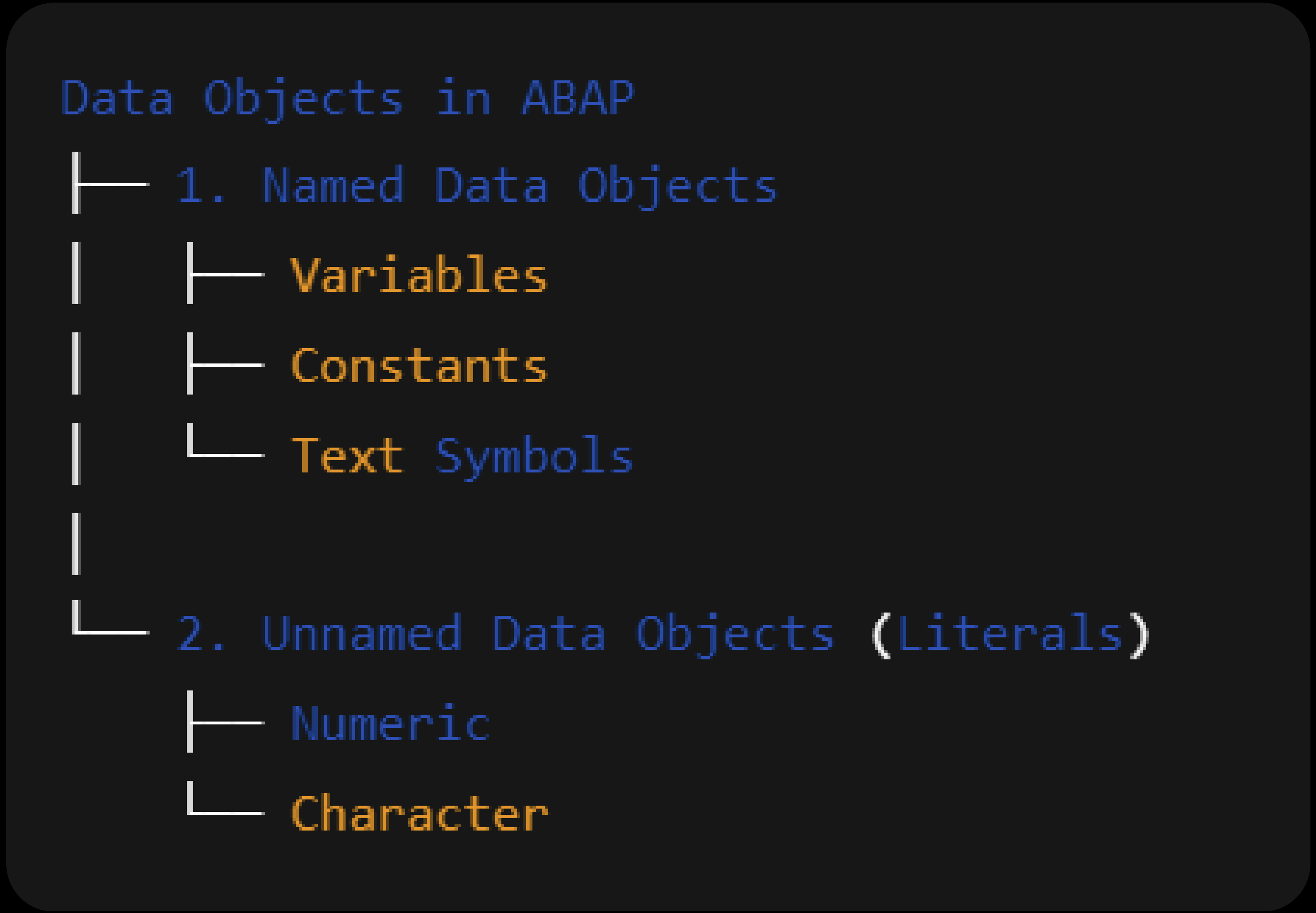
The data type defines the technical attributes of the data object.

The data type does not use any memory space. Technical attributes -> type, length & etc

- Elementary
- Complex
- Reference

```
ABAP Data Types
├── 1. Elementary Types
│   ├── Numeric Types
│   │   ├── I (Integer)
│   │   ├── F (Floating Point)
│   │   ├── P (Packed Decimal)
│   ├── Character Types
│   │   ├── C (Character)
│   │   ├── N (Numeric Text)
│   │   ├── STRING (Variable-length)
│   ├── Date/Time Types
│   │   ├── D (Date)
│   │   ├── T (Time)
│   └── Other Types
│       ├── X (Hexadecimal)
│       ├── XSTRING (Variable-length Hex)
│       ├── CLNT (Client)
│       └── LANG (Language Key)
│
├── 2. Complex Types
│   ├── Structures
│   │   └── Custom or predefined group of fields (like a record)
│   ├── Internal Tables
│   │   └── Temporary tables in memory (Standard, Sorted, Hashed)
│   └── Table Types
│       └── DDIC-defined template for internal tables
│
└── 3. Reference Types
    ├── Data References
    │   └── Ref to a data object (e.g., `DATA REF TO i`)
    └── Object References
        └── Ref to a class instance (e.g., `DATA obj TYPE REF TO <class>`)
```

Data Objects



```
Data Objects in ABAP
├── 1. Named Data Objects
|   ├── Variables
|   ├── Constants
|   └── Text Symbols
|
└── 2. Unnamed Data Objects (Literals)
    ├── Numeric
    └── Character
```

- A data object is a part of the repository whose content can be addressed and interpreted by the program.
- All data objects must be declared in the ABAP program and are not persistent, meaning that they only exist while the program is being executed.
- Before you can process persistent data (such as data from a database table or from a sequential file), you must read it into data objects first.

Data Objects

LITERALS:

- A literal is a fixed value directly written into the source code.
- Represents constant values that do not change during execution.
- Embedded within code and do not have a name or identifier.
- Examples:
  - Numeric: 10, -5
  - Character: 'Hello'

NAMED DATA OBJECTS:

- Declared with a specific name and data type.
- Includes: variables, constants, fields, structures, internal tables, database tables.
- Referred to by their identifiers in the program.
- Used to store and manipulate data dynamically at runtime.
- Unlike literals, can change values and are declared separately from the logic.

MASTER DATA VS TRANSACTIONAL DATA

Master data:

- Master Data is the primary, mostly non-frequently changeable data of an organization.
- It represents the core information essential for business operations.
- Examples: Customer Name, Customer Address, Product Information, Vendor Details

Transactional data:

- Transactional data refers to the dynamic data generated from day-to-day operations of an organization. It records business events or transactions and changes frequently over time.
- Examples: Sales Order, Purchase Order, Invoice, Payment

SHORTCUTS and T-CODES

Short-cuts:

- Ctrl + F2 - Syntax check
- Ctrl + F3 - Activate
- F9 - Run and Opens console
- F4 - Search help
- F5 - Debugger single step
- Double tap next to line number - debugger
- F8 - Display table from schema
- Ctrl + Space - Autofill

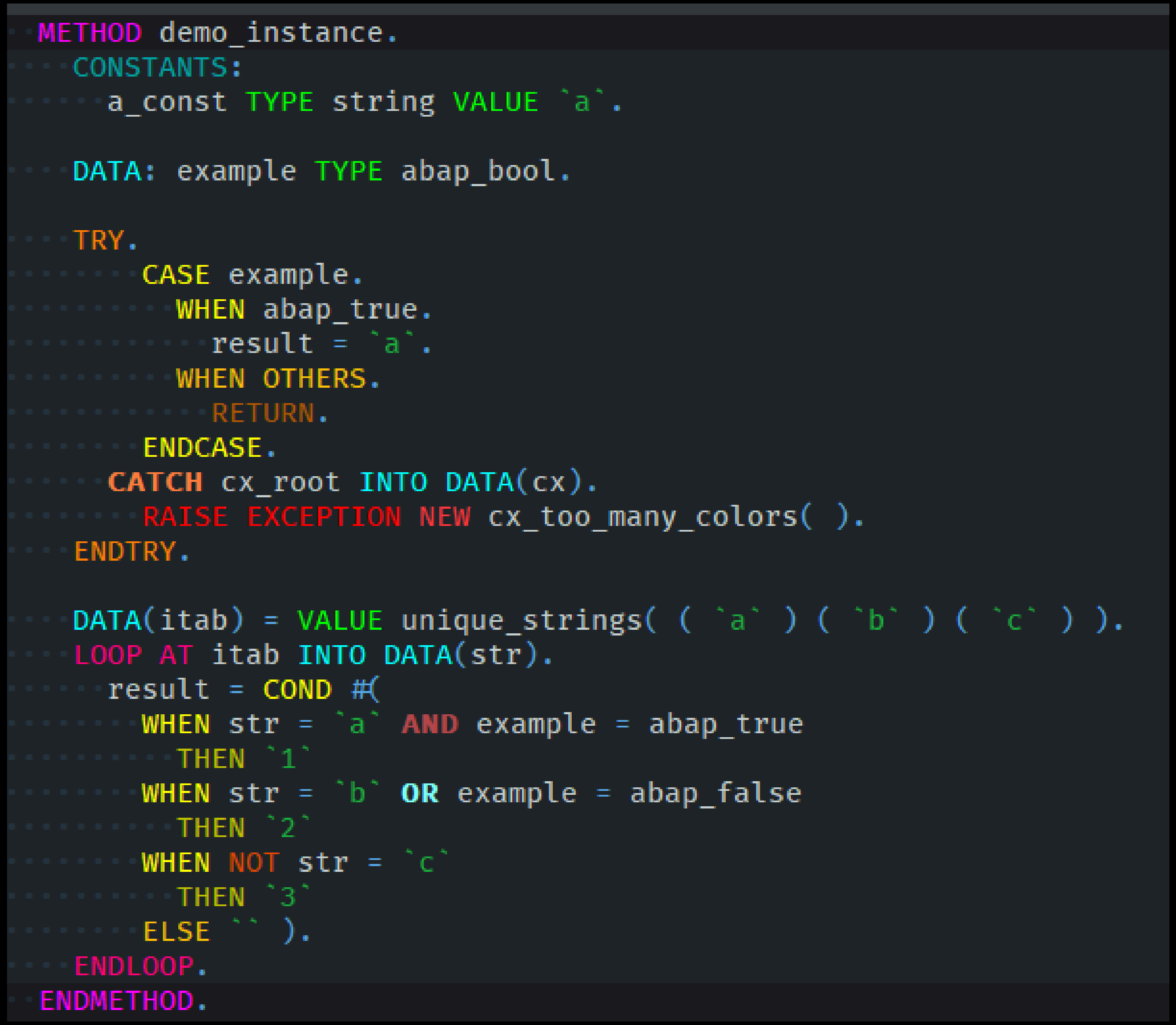
T-Codes:

- SE11 - ABAP Data Dictionary
- SE38 - ABAP Editor
- SE69 - To view the data display of entires
- SE14 - Managing Databases after changes made in SE11

SYNTAX

NOTE:

- ABAP is not case sensitive
- Most of the time issue is either "." or space.
- Error messages will not have any relation to the actual error most of the times.



```
METHOD demo_instance.
  CONSTANTS:
    a_const TYPE string VALUE `a`.

  DATA: example TYPE abap_bool.

  TRY.
      CASE example.
        WHEN abap_true.
          result = `a`.
        WHEN OTHERS.
          RETURN.
      ENDCASE.
    CATCH cx_root INTO DATA(cx).
      RAISE EXCEPTION NEW cx_too_many_colors( ).
  ENDTRY.

  DATA(itab) = VALUE unique_strings( ( `a` ) ( `b` ) ( `c` ) ).
  LOOP AT itab INTO DATA(str).
    result = COND #(
      WHEN str = `a` AND example = abap_true
        THEN `1`
      WHEN str = `b` OR example = abap_false
        THEN `2`
      WHEN NOT str = `c`
        THEN `3`
      ELSE `` ).
  ENDLOOP.
ENDMETHOD.
```

HOW TO GET STARTED?

1) Start by creating a new ‘ABAP Cloud Project’

2) Create a Package.

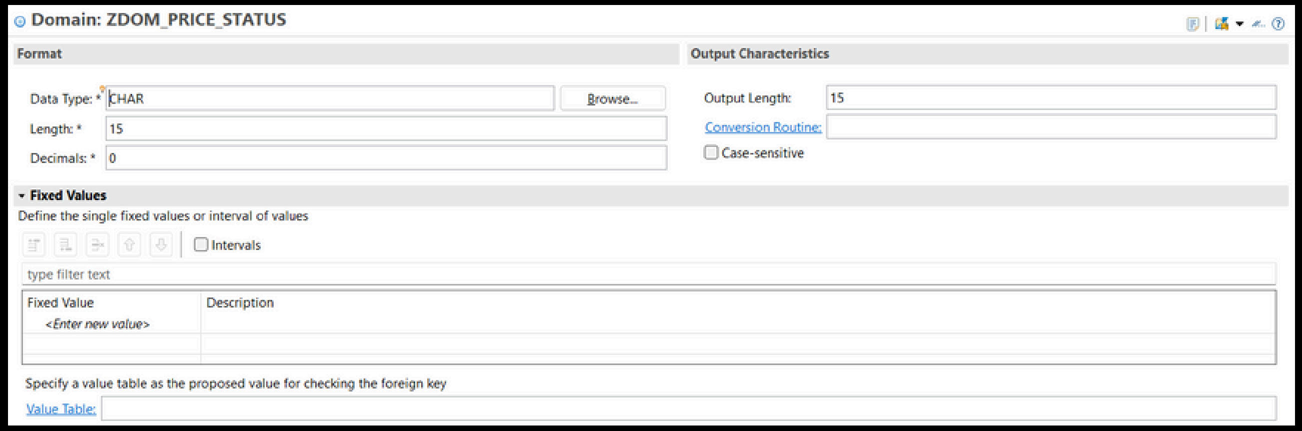
3) Create a Class - must specify interface(if_oo_adt_classrun~main) also

4) Within the package you can now create DDIC components and work with them

Creating DDIC Components

For any DDIC component - Ctrl + N then select which component

DOMAIN: Specify Data type and length



Domain: ZDOM_PRICE_STATUS

Format

Data Type: *	CHAR	Browse...
Length: *	15	
Decimals: *	0	

Output Characteristics

Output Length:	15
Conversion Routine:	

☐ Case-sensitive

Fixed Values

Define the single fixed values or interval of values

☐ Intervals

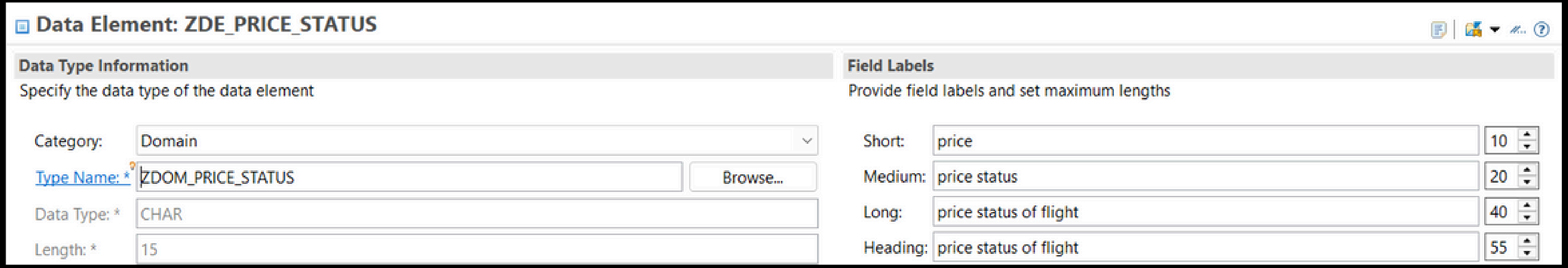
type filter text

Fixed Value	Description
<Enter new value>	

Specify a value table as the proposed value for checking the foreign key

Value Table:

DATA ELEMENT: Specify Datatype and Domain



Data Element: ZDE_PRICE_STATUS

Data Type Information

Specify the data type of the data element

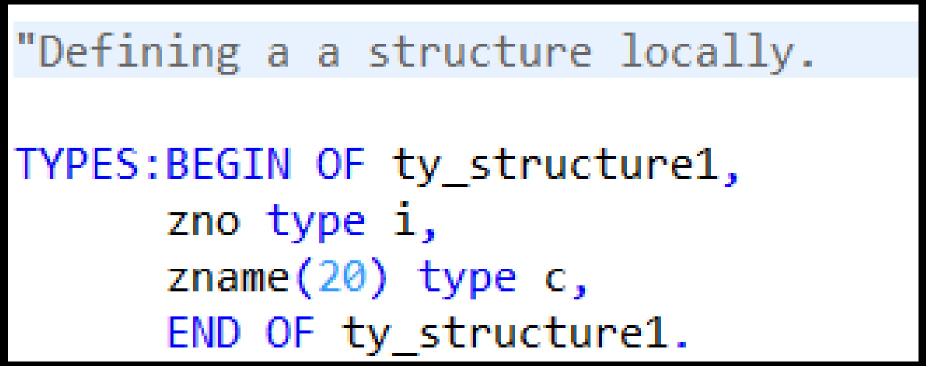
Category:	Domain	
Type Name: *	ZDOM_PRICE_STATUS	Browse...
Data Type: *	CHAR	
Length: *	15	

Field Labels

Provide field labels and set maximum lengths

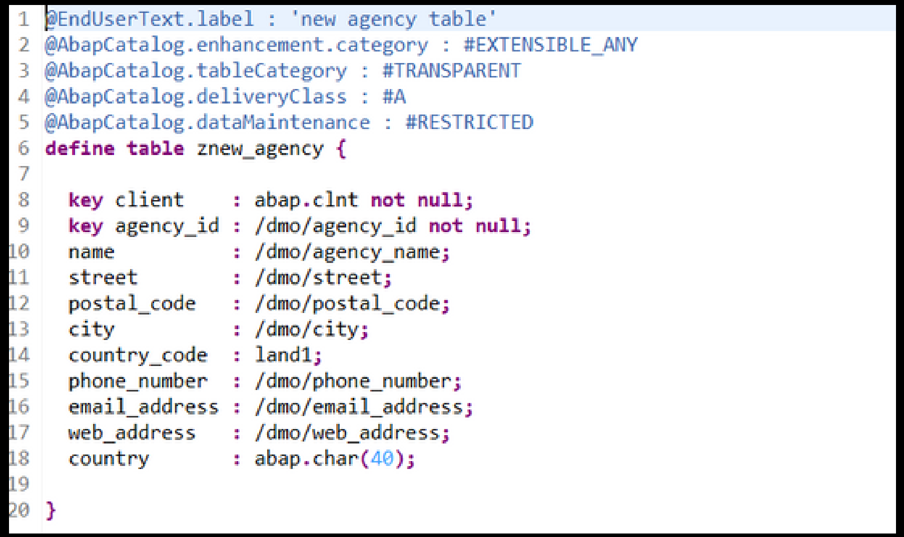
Short:	price	10
Medium:	price status	20
Long:	price status of flight	40
Heading:	price status of flight	55

STRUCTURE: Define schema text



```
"Defining a a structure locally.

TYPES:BEGIN OF ty_structure1,
      zno type i,
      zname(20) type c,
      END OF ty_structure1.
```

DATABASE TABLE: Define schema

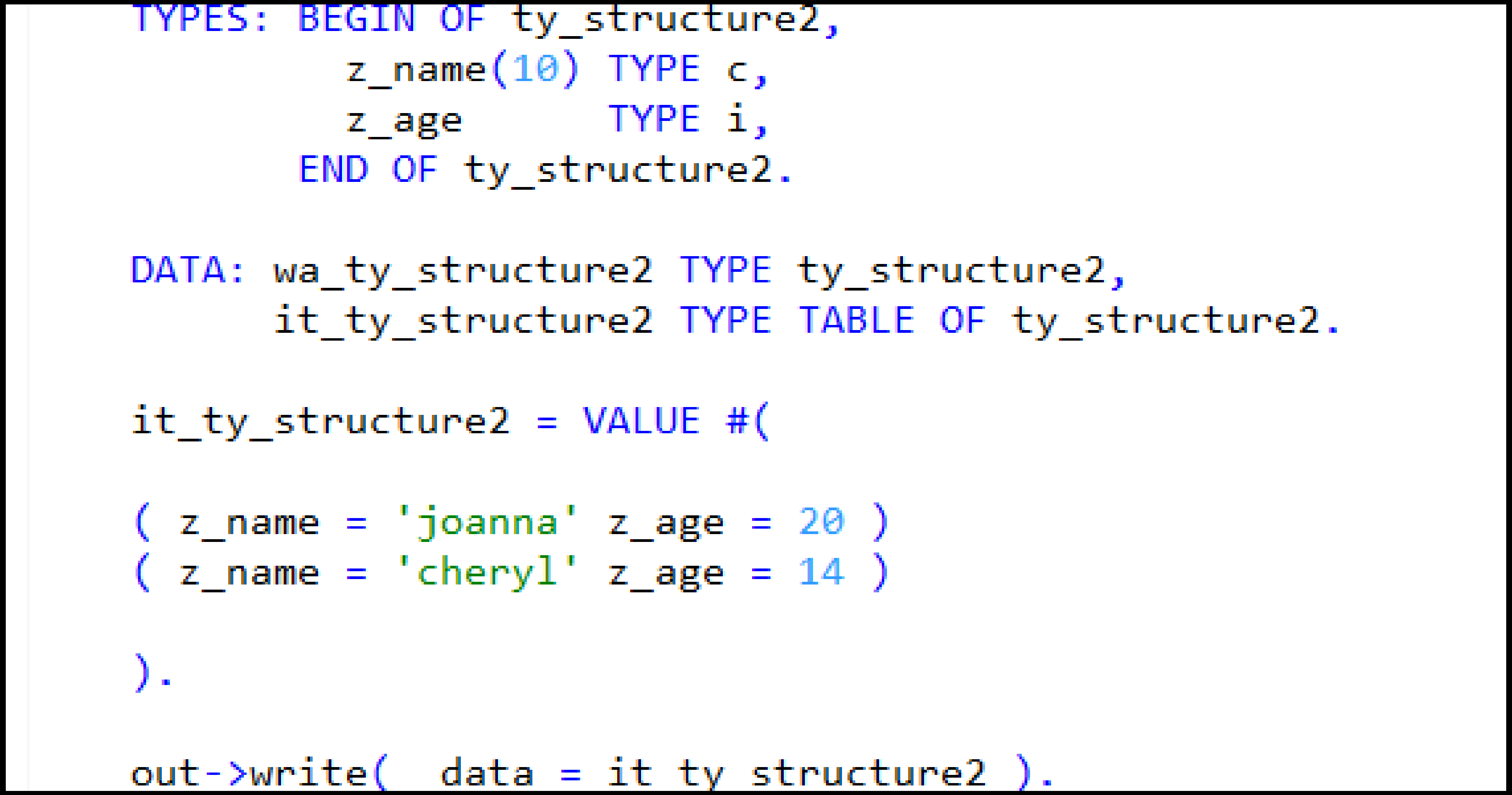


```
@EndUserText.label : 'new agency table'
@AbapCatalog.enhancement.category : #EXTENSIBLE_ANY
@AbapCatalog.tableCategory : #TRANSPARENT
@AbapCatalog.deliveryClass : #A
@AbapCatalog.dataMaintenance : #RESTRICTED
define table znew_agency {

  key client     : abap.clnt not null;
  key agency_id  : /dmo/agency_id not null;
  name           : /dmo/agency_name;
  street         : /dmo/street;
  postal_code    : /dmo/postal_code;
  city           : /dmo/city;
  country_code   : land1;
  phone_number   : /dmo/phone_number;
  email_address  : /dmo/email_address;
  web_address    : /dmo/web_address;
  country        : abap.char(40);

}
```

Creating DDIC Components

For any DDIC component - Ctrl + N then select which component

TABLE TYPE: Creating from structures



```
TYPES: BEGIN OF ty_structure2,
         z_name(10) TYPE c,
         z_age      TYPE i,
       END OF ty_structure2.

DATA: wa_ty_structure2 TYPE ty_structure2,
      it_ty_structure2 TYPE TABLE OF ty_structure2.

it_ty_structure2 = VALUE #(

( z_name = 'joanna' z_age = 20 )
( z_name = 'cheryl' z_age = 14 )

).

out->write(  data = it_ty_structure2 ).
```

CODE SYNTAX - GENERAL

- **Initialising a variable:** DATA variable name TYPE datatype.
  EXAMPLE: DATA lv_variable TYPE i.

- **CHAIN OPERATOR →** ':' used to apply a keyword to multiple statements.
  EXAMPLE → DATA : variable_1 TYPE i,
  variable_2 TYPE c.

- **TYPES:** used to create blueprints such as structures and table types

Conditional Statements

Conditional statements in abap are fundamental control structures that allow your program to execute different blocks of code based on whether certain conditions are met

1. IF - ELSE

- Executes a block of code if given condition is true.
- Executes a different block of code if the given condition is false.

SYNTAX:

```
IF <condition>.
    Statement if condition is true.
ELSE.
     Statement if condition is false.
ENDIF.
```

2. IF - ELSEIF - ELSE

- Evaluates multiple conditions sequentially.
- If first condition is true, executes the corresponding block of code.
- If second condition is true, executes the second bock of code and so on.
- Executes the last block of code if none of the conditions given is satisfied.

SYNTAX:

```
IF <condition1>.
    Statement if condition 1 is true.
ELSEIF <condition 2>.
    Statement if condition 2 is true.
ELSE.
     Statement if  none of the condition
      is true.
ENDIF.
```

3.CASE

- Considers a particular operand and executes statements according to the value of that operand.
- Once a case is matched the program automatically comes out of the case block.

SYNTAX:

```
CASE <operand>.
WHEN <value 1>.
        Statement 1.
WHEN <value 2>.
         Statement 2.
WHEN others.
         Stattement 3.
ENDCASE.
```

Conditional Expressions

Conditional expressions allow you to assign values based on conditions in a compact and readable way, without using full control structures like IF or CASE.

1. COND

- COND is a conditional expression introduced in ABAP 7.40+.
- It returns a value of the specified type, based on the first WHEN condition that evaluates to true.
- If none of the conditions are met, the ELSE clause is executed.
- let_exp allows defining temporary variables used in the result.

SYNTAX:

```
COND type(
  [let_exp]
  WHEN log_exp1 THEN [let_exp] result1
  [WHEN log_exp2 THEN [let_exp] result2]
  ...
  [ELSE [let_exp] resultn]
)
```

2. SWITCH

- SWITCH is a case-based expression introduced in ABAP 7.40+.
- It compares an operand against multiple values.
- Returns the result for the first matching value.
- If no match is found, the ELSE clause is returned.

SYNTAX:

```
SWITCH type(
  operand
  WHEN value1 THEN result1
  [WHEN value2 THEN result2]
  ...
  [ELSE resultn]
)
```

LOOPS IN ABAP

1) DO LOOP

```
DO [n TIMES].
  " loop body
ENDDO.
```

2) LOOP AT

```
LOOP AT <itab> INTO <wa>.
  " loop body
ENDLOOP.
```

3) WHILE LOOP

```
WHILE <condition>.
  " loop body
ENDWHILE.
```

- CONTINUE – Skip to the next iteration of a loop
- EXIT – Terminate the loop immediately

SYSTEM VARIABLES:

1) sy-subrc

- Stores the return code of the last ABAP statement.
- Value:
  - 0 → Operation successful
  - ≠ 0 → Operation failed or condition not met
- Commonly used after SELECT, READ, INSERT, DELETE, etc.

2) sy-index

- Current loop iteration counter
- Used in: DO, WHILE, LOOP AT (with index)

3) sy-tabix

- Purpose: Holds the current row number in internal tables
- Useful for: Knowing the position of the row found
  or process**ed**

SQL IN ABAP

- There are 2 types: open and native
- We are currently dealing with open sql.
- OpenSQL allows access to the ABAP Data Dictionary without checking which database platform the R3 system uses.
- It uses ABAP statements to operate upon the R3 system's central database.

SQL SYNTAX

KEYWORDS LEARNT: INSERT, SELECT, WHERE, JOINS

SELECT + WHERE :

```
SELECT * FROM <db_table>
  INTO TABLE @DATA(it_data)
  WHERE <condition>.
```

→ use SELECT SINGLE to select a single row
→ in this case using WHERE is a must

INSERT:

```
INSERT <db_table> VALUES @wa_data.
```

JOINS

- Joins in ABAP are used to combine rows from two or more database tables, views or internal tables based on the related columns between them.

- This is done using the **SELECT** statement in Open SQL.

TYPES OF JOINS IN ABAP OPEN SQL:

- **Inner join** : Returns only the rows that match in both the tables based on the 'ON' condition.
- **Left Outer Join** : Returns only unmatched data from the left table.
- **Right Outer Join** : Returns only unmatched data from the right table

JOINS SYNTAX

```
SELECT a~field1, b~field2
  FROM table1 AS a
  INNER JOIN table2 AS b
    ON a~common_field = b~common_field
  INTO TABLE @DATA(result).
```

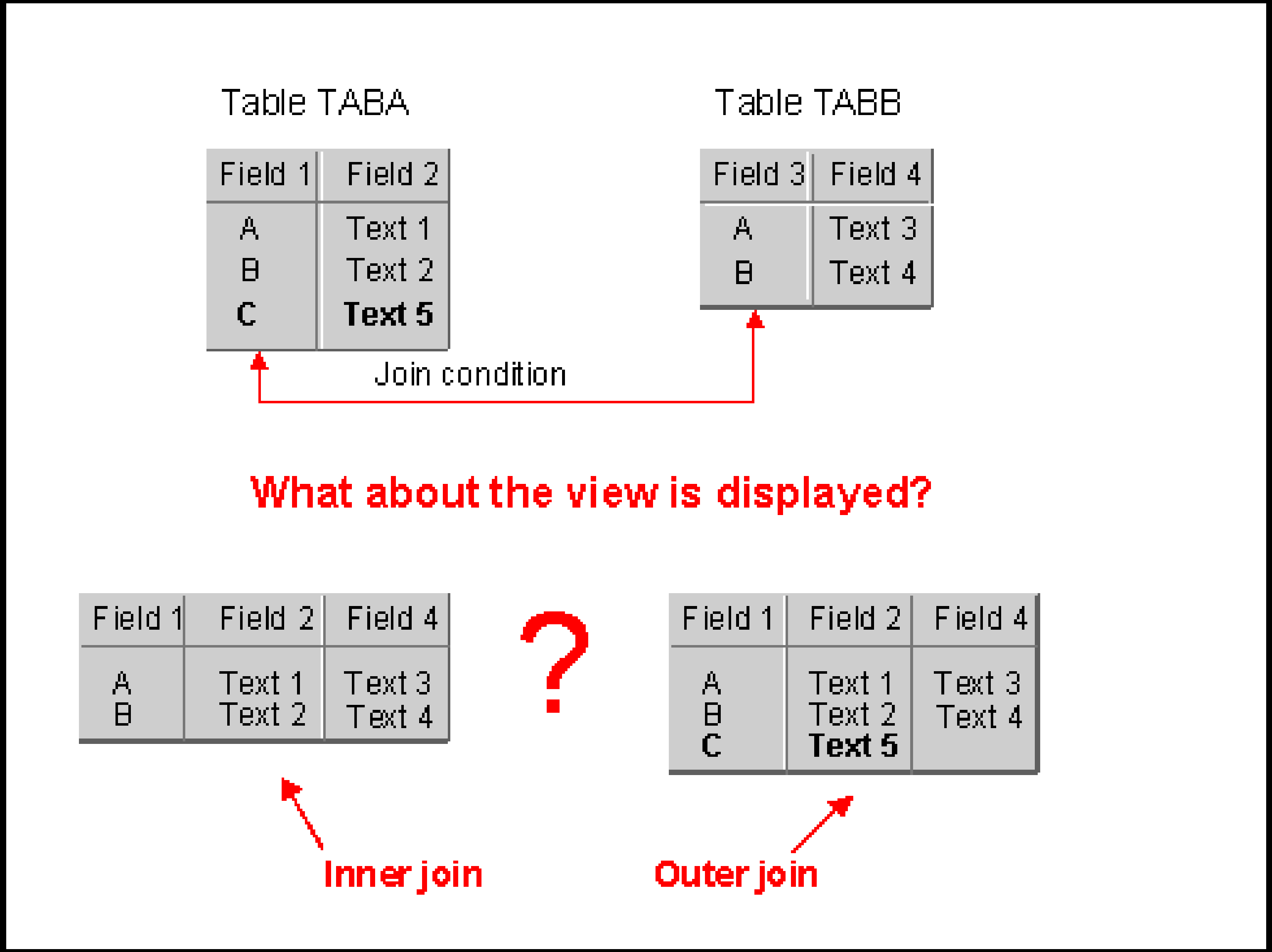


Table TABA

Field 1	Field 2
A	Text 1
B	Text 2
C	**Text 5**

Table TABB

Field 3	Field 4
A	Text 3
B	Text 4

Inner join

Field 1	Field 2	Field 4
A	Text 1	Text 3
B	Text 2	Text 4

Outer join

Field 1	Field 2	Field 4
A	Text 1	Text 3
B	Text 2	Text 4
C	**Text 5**	

FUNCTION GROUP

- A function group is a container that has a collection of function modules in ABAP.
- It are also referred to as ‘Function Pool’.
- It allows the reuse of code across different applications within the SAP system.
- When function modules need to be created, function groups are created first, then the function modules are created and assigned to a particular function group.

FUNCTION MODULE

- A function module is a reusable piece of ABAP code that is used to perform a specific task.
- It can be used in different ABAP programs across the SAP system.
- Function modules are stored in a central function library and can be called by any ABAP program by referring to the name of the function module.
- When creating a function module each function module is assigned to a function group.
- They are called in a class using '**CALL FUNCTION**' statement.

SYNTAX:

CALL FUNCTION 'Function_module_name'.

PARAMETERS OF FUNCTION MODULE

There are four types of parameters in ABAP Function Modules:

- **Importing parameters :** Used to pass data from the calling program into the function module.

  SYNTAX:
  ```
  IMPORTING
          VALUE (var_name) TYPE datatype
  ```

- **Exporting parameters :** Used to pass data from the function module into the calling program.

  SYNTAX:
  ```
  EXPORTING
          VALUE(var_name) TYPE datatype
  ```

PARAMETERS OF FUNCTION MODULE

- **Changing parameters** : Used to pass data into the function module, allow it to modify the data and then pass it back to the calling program.

  SYNTAX :
  ```
  CHANGING
          var_name TYPE datatype
  ```

- **Table parameters** : Used to pass internal tables between a calling program and a function module.

  SYNTAX:
  ```
  var_name LIKE wa_name
   (or)
   var_name TYPE table_type
  ```

THANK YOU!